901画像処理ライブラリ
FVL基本統合SDK/902(FVL-SDK-ITG/902)
903画像処理ライブラリ
FVL基本SDK/904(FVL-SDK-BSC/904)
FVL基本SDK/LNX(FVL-SDK-BSC/LNX)

90X
基本ライブラリ説明書
Vol.2

☆第 14 版☆

（株）ファースト

御注意

◎本書の内容の一部または全部を無断で転載することは固くお断りします。
◎本書の内容について、将来改良を目的に予告なしに変更することがあります。
◎MCP960™は、Mentor Graphics Corporationの商標です。
◎Watcom C™は、Sybase Inc.とその関連会社の商標です。
◎Pentium™は、Intel Corporationの商標です。
◎Green Hillsロゴ、およびMULTIは、Green Hills Software,Inc.の商標です。

はしがき

この説明書は、

901画像処理ライブラリ　Ver.6.50

FVL基本統合SDK/902(FVL-SDK-ITG/902) Ver.5.10

903画像処理ライブラリ　Ver.2.00

FVL基本SDK/904(FVL-SDK-BSC/904) Ver.2.00

FVL基本SDK/LNX Ver.3.00

に対応しています。

本書は、90X シリーズ用ユーザプログラム作成時に使用できる、「基本ライブラリ」について記載したものです。

なお、この他に 90X シリーズライブラリとしては、下記のようなものがあります。

◎90X　基本ライブラリ(Vol.1,Vol.3)説明書

◎90X　2 値画像ライブラリ説明書

◎90X　濃淡画像ライブラリ説明書

◎90X　ビジョン･ツール･ライブラリ説明書

◎90X　キャリパーライブラリ説明書

◎902　高分解能センサライブラリ説明書

ユーザプログラムの開発方法その他につきましては

◎FAST　Vision　Library プログラマーズガイド

◎90X　操作説明書

◎FVL/LNX　操作説明書

をご参照ください。

[注 1] 901 用のコンパイラ MCP960(960　C クロスコンパイラ)のバージョンは 2.0 以上です。

[注 2] 902 用のコンパイラ Watcom C のバージョンは 10.5 以上です。

[注 3] 903,904 用のコンパイラ Green Hills C および MULTI のバージョンは 1.8.9 以上です。

目　　次

１．概　　要 ・・・

目　　次

ビデオウィンドウの描画(Lib_vout_draw_video_window)----------80
ビデオウィンドウの消去(Lib_vout_erase_video_window)----------81
ビデオウィンドウオプションの取得(Lib_vout_get_option)----------82
ビデオウィンドウオプションの変更(Lib_vout_set_option)----------83

3．メモリ領域割当ライブラリ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・85
濃淡画像メモリの確保(Lib_alloc_gray_memory)----------88
2値画像メモリの確保(Lib_alloc_bin_memory)----------90
濃淡画像メモリのベース・ポインタ通知(Lib_adrs_gray_memory)----------92
2値画像メモリのベース・ポインタ通知(Lib_adrs_bin_memory)----------93
濃淡画像メモリの解放(Lib_free_gray_memory)----------95
2値画像メモリの解放(Lib_free_bin_memory)----------97
濃淡画像メモリにおけるカレントメモリ番号.の参照(Lib_get_gray_memory)----------99
2値画像メモリにおけるカレントメモリ番号.の参照(Lib_get_bin_memory)----------100
グレイ&2値のカレントメモリ切り換え(Lib_change_memory)----------101
濃淡画像メモリの切り換え(Lib_change_gray_memory)----------102
2値画像メモリの切り換え(Lib_change_bin_memory)----------103
画像メモリの大きさ通知　横方向画素数(Lib_get_fx_size)----------104
画像メモリの大きさ通知　縦方向画素数(Lib_get_fy_size)----------106
フレーム・バッファの大きさ通知　横方向画素数(Lib_get_dx_size)----------108
フレーム・バッファの大きさ通知　縦方向画素数(Lib_get_dy_size)----------109
メモリブロックの割当(Lib_malloc)----------110
メモリブロックの割当Ⅱ(Lib_mlalloc)----------111
メモリブロックの解放(Lib_free)----------112
メモリブロックの解放Ⅱ(Lib_lfree)----------113
メモリタイプの取得(Lib_get_memory_type)----------114
拡張画像メモリ時用のワークメモリアドレスの取得(Lib_get_adrs_work_memory)----------115
濃淡フレーム・バッファへの矩形領域画像転送(Lib_gray_transmit)----------117
連続画像バッファの画像メモリへの転送(Lib_buff_to_memory)----------118
濃淡画像メモリの大きさ取得(Lib_get_gray_memory_size)----------119
2値画像メモリの大きさ取得(Lib_get_bin_memory_size)----------120

4．メモリ操作ライブラリ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・121
メモリのクリア(Lib_memory_clear)----------124
2値画像メモリクリア(Lib_bin_memory_cls)----------125
濃淡画像メモリクリア(Lib_gray_memory_cls)----------126

5．画像計測ライブラリ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・127
ヒストグラムの計算(Lib_histogram)----------130
ヒストグラムの計算(1画素飛び)(Lib_histogram_step)----------132
ヒストグラムデータの表示(Lib_display_histogram)----------134
Aスコープ表示(Lib_ascope)----------137
濃度値鳥瞰図描画(Lib_g3dwire)----------139
拡張画像メモリ時用Aスコープ表示(Lib_ex_ascope)----------144

6．標準入出力ライブラリ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・147
標準入出力のタイムアウト時間発生時間の設定(Lib_set_io_timeout)----------150
標準入出力のタイムアウト時間発生時間の参照(Lib_get_io_timeout)----------152
標準入出力の開始(Lib_open_io)----------154
標準入出力の終了(Lib_close_io)----------156

目　次

標準入力から１文字入力(Lib_getchar) ---- 158
標準出力へ１文字出力(Lib_putchar) ---- 159
標準入力から文字列入力(Lib_gets) ---- 160
標準出力から文字列出力(Lib_puts) ---- 162
書式つき出力(Lib_printf) ---- 163
文字列への書式つき出力(Lib_sprintf) ---- 166
書式つき入力(Lib_scanf) ---- 167
文字列からの書式つき入力(Lib_sscanf) ---- 170
標準入出力のエラー情報参照(Lib_get_io_error) ---- 171
標準入出力用バッファの入力チェック(Lib_io_buff_check) ---- 172

7．RS232C ライブラリ ・・・

目　　次

9.5　通信手順 ····· 235

10．C-LINK ライブラリ ····· 237
C-LINK のオープン(Lib_open_clink) ----- 240
C-LINK のクローズ(Lib_close_clink) ----- 242
C-LINK データ受信(Lib_gets_clink) ----- 243
C-LINK1 文字受信(Lib_getc_clink) ----- 244
C-LINK データ送信《Lib_puts_clink》 ----- 246
C-LINK1 文字送信(Lib_putc_clink) ----- 247

11．パラレル・リンクライブラリ ····· 249
パラレル・リンクの初期化(Lib_open_plink) ----- 252
パラレル・リンクの終了(Lib_close_plink) ----- 254
受信ステータスの取得(Lib_receive_status_plink) ----- 255
文字列を受信(Lib_gets_plink) ----- 256
1 文字受信(Lib_getc_plink) ----- 257
文字列を送信(Lib_puts_plink) ----- 259
1 文字送信(Lib_putc_plink) ----- 260

12．E-LINK ライブラリ ····· 261
E-LINK オープン(Lib_open_elink) ----- 265
E-LINK クローズ(Lib_close_elink) ----- 266
E-LINK サーバ開始(Lib_elink_server) ----- 267
E-LINK クライアント開始(Lib_elink_client) ----- 268
E-LINK 状態監視(Lib_elink_status) ----- 270
E-LINK 受信(Lib_elink_receive) ----- 271
E-LINK 送信(Lib_elink_send) ----- 272
E-LINK 情報取得(Lib_elink_get_client) ----- 273
E-LINK 切断(Lib_elink_disconnect) ----- 274
12.1　ユーザープログラム作成例 ····· 275

13．外部入出力端子ライブラリ ····· 279
外部入力端子(4 点)状態参照(Lib_external_input) ----- 282
外部出力端子(4 点)状態設定(Lib_external_output) ----- 283
アラート出力制御(Lib_alert_output) ----- 284
トリガ割り込み関数設定(Lib_external_trigger) ----- 285
外部出力端子 露光期間出力設定(Lib_external_set_exposure_output) ----- 286
外部出力端子 露光期間出力設定値 参照(Lib_external_get_exposure_output) ----- 288

14．時刻サービスライブラリ ····· 291
実時間クロック初期化(Lib_strtclk) ----- 294
実時間クロック読み取り(Lib_readclk) ----- 295
インタバルタイマ設定(Lib_strtetc) ----- 296
インタバルタイマ状態検知(Lib_sensetc) ----- 297
インタバルタイマ取消(Lib_cancetc) ----- 298
タイムディレイ(Lib_time_delay) ----- 299
実時間クロックスタート(微細クロックモード)(Lib_strtclk_count) ----- 300
実時間クロック読み取り(微細クロックモード)(Lib_readclk_count) ----- 301
実時間クロックの読み取り表示(微細クロックモード)(Lib_time_disp) ----- 302

実時間クロック停止(微細クロックモード)(Lib_stopclk_count)---303
時刻の取得(Lib_get_time)---304
時刻の設定(Lib_set_time)---305
日付の取得(Lib_get_date)---306
日付の設定(Lib_set_date)---308
電池の残容量確認(Lib_check_battery)---309
実時間読み取り(Lib_xreadclk)---310

15．数値演算ライブラリ・・・311
正弦関数(Lib_sinfunc)---314
余弦関数(Lib_cosfunc)---315
平方根関数(Lib_sqrt32)---316
逆正接関数(Lib_atan360)---317

16．特殊計測・計算処理ライブラリ・・・319
最小自乗法直線近似(Lib_calcline)---322
最小自乗法円近似(Lib_fcalcircl)---325
最小自乗法楕円近似(Lib_fcalelps)---328
最小自乗法 2 次曲線近似(Lib_fcalquad)---331
2 次曲線描画(Lib_drawquad)---332

17．アフィン変換ライブラリ・・・333
画像の回転(Lib_rotation)---335
画像の回転Ⅱ(Lib_xrotation)---337
画像の拡大縮小(Lib_scale)---339
画像の拡大縮小Ⅱ(Lib_xscale)---341
画像の 2 倍拡大(Lib_enlarge)---343
画像の 1/2 縮小(Lib_reduce)---344
画像の平行移動(Lib_shift)---345
扇型エリアを矩形エリアに変換(Lib_pol_to_rec)---347

18．座標変換(キャリブレーション)ライブラリ・・・349
18.1　前提条件・・・352
キャリブレーションの開始(Lib_calibstart)---353
対応点位置設定(Lib_calibposition)---355
キャリブレーションの終了(Lib_calibfinish)---356
座標変換実施(Lib_coordconv)---358
拡張キャリブレーションの開始(Lib_xcalibstart)---360
拡張キャリブレーションの対応点位置設定(Lib_xcalibposition)---362
拡張キャリブレーションの終了(Lib_xcalibfinish)---363
拡張座標変換の実施(Lib_xcoordconv)---364

19．カーソル関連ライブラリ・・・365
カーソル初期化(Lib_init_cursor)---368
カーソル表示(Lib_draw_cursor)---370
カーソル消去(Lib_erase_cursor)---371
カーソル移動(Lib_move_cursor)---372
カーソル表示モード取得(Lib_get_cursor_mode)---373
カーソル情報取得(Lib_see_current_position)---374

目　　次

カーソル情報取得Ⅱ(Lib_look_current_position)----------376
カーソル情報取得(現在状況取得)(Lib_see_current_position_real)----------379
確定キー待ち機能付きカーソル情報取得(Lib_get_current_position)----------380
カーソル移動速度率設定(Lib_set_cursor_rate)----------383
カーソル座標取得(Lib_get_current_geometry)----------384

付録. 各ライブラリの処理速度一覧 · 385

1. 概　　要

本書は基本機能をライブラリの形で供給するほかに、90X シリーズの保有するハードウェアの制御サービス、その他を供給するものです。
本書(Vol.2)では、各ライブラリについて、下記の 18 区分に分けて記載します。

Vol.1

1. ビデオ入力制御ライブラリ

Vol.2

1. グラフィックライブラリ
2. メモリ領域割当ライブラリ
3. メモリ操作ライブラリ
4. 画像計測ライブラリ
5. 標準入出力ライブラリ
6. RS232C ライブラリ
7. C-NET 基本ライブラリ
8. C-NET 応用ライブラリ
9. C-LINK ライブラリ
10. パラレル･リンクライブラリ
11. E-LINK ライブラリ
12. 外部入出力端子ライブラリ
13. 時刻サービスライブラリ
14. 数値演算ライブラリ
15. 特殊計測･計算処理ライブラリ
16. アフィン変換ライブラリ
17. 座標変換(キャリブレーション)ライブラリ
18. カーソル関連ライブラリ

Vol.3

1. パッド関連ライブラリ
2. メニュー･バー関連ライブラリ
3. リスト(表形式)関連ライブラリ
4. 照明制御ライブラリ
5. ファイルシステムライブラリ
6. システム制御ライブラリ
7. システムパラメータ関連ライブラリ
8. フォトカプラ絶縁型 I/O 拡張ライブラリ

なお、各ライブラリ関数の表記

901	902	903	904	FVL/LNX	高分解能

は、901,902,903,904 及び高分解能カメラ使用時にライブラリ関数が使用できるかどうかを表すもので

○ ・・・・・・・・・・ 使用可能
△ ・・・・・・・・・・ 一部使用可能
空白・・・・・・・・・・ 使用不可

となっております。
△の場合は、必ず留意事項を参照してください。

2. グラフィックライブラリ

本ライブラリはモニタ-TV に文字、線を表示あるいは画像メモリに線を作画するものです。

Lib_set_font_type

機　能

ANK 表示文字のフォント･タイプの設定

形　式

```
#include "f_graph.h"
int Lib_set_font_type( int type );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

ANK 表示文字のフォント・タイプを変更します。
種類は 2 種類あります。

① ***type*** はフォント・タイプです。

値	定　　数	意　　味
0	FONT_TYPE1	フォント･タイプ 1 を表示します。 (デフォルト)
1	FONT_TYPE2	フォント･タイプ 2 を表示します。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

フォント･タイプをタイプ 2 に変更します。

```
#include "f_graph.h"

void set_font()
{
    Lib_set_font_type( FONT_TYPE2 );

    Lib_chrdisp( 10, 10, "Font Type2" );
}
```

留意事項

○ システムパラメータの「モニタ TV 表示オプション」が指定されたパラメータにより、更新されます。

Lib_get_font_type

機　能

ANK 表示文字のフォント･タイプの参照

形　式

```
#include "f_graph.h"
int Lib_get_font_type( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

ANK 表示文字のフォントを参照します。
フォント種類は 2 種類あります。

戻り値

フォントタイプ

値	定　　数	意　　味
0	FONT_TYPE1	フォント･タイプ 1 を使用中です。 (デフォルト)
1	FONT_TYPE2	フォント･タイプ 2 を使用中です。

例

フォント･タイプがタイプ 2 でない場合はタイプ 2 に変更します。

```
#include "f_graph.h"

void set_font()
{
    if ( FONT_TYPE2 !=  Lib_get_font_type())
         Lib_set_font_type( FONT_TYPE2 );

    Lib_chrdisp( 10, 10, "Font Type2" );
}
```

留意事項

ありません。

Lib_set_cursor_shape

機　能

カーソル形状の登録

形　式

```
#include "f_graph.h"
void Lib_set_cursor_shape ( int type );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソルの形状を変更します。

① ***type*** は種類です。

値	定　　数	意　　味
1	CROSS_TYPE	クロス形状を使用します。
2	HOURGLASS_TYPE	砂時計形状を使用します。
3	HAND_TYPE	手袋形状を使用します。
4	MOVE_TYPE	4 方向への移動形状を使用します。
-1	ARROW_TYPE	矢印形状使用(標準)を使用します。

戻り値

ありません。

例

```
#include "f_graph.h"

void cursor()
{
    /*カーサ表示モードの変更*/
    Lib_set_cursor_shape ( HOURGLASS_TYPE );

    /*処理を行う*/

    /*カーサ表示モードの復帰*/
    Lib_set_cursor_shape ( ARROW_TYPE );
}
```

留意事項

ありません。

FVL/LNX

○ 現在のところ本関数は何も実行しません。

Lib_kanjishift

機　能

シフト JIS 漢字コード表示

形　式

```
#include "f_graph.h"
int Lib_kanjishift( int rvs_f, int dbl_f, int step_f,
                    int xpos, int ypos, char *strings );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字列(ANK と旧 JIS 第 I 水準のシフト JIS)を文字フレームバッファの指定位置に表示します。
フォントサイズは
　☆ANK‥‥‥

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

文字列「テスト表示」を反転、標準サイズで表示します。

```
#include "f_graph.h"

void disp_test()
{
    Lib_kanjishift( REVERSE_FONT, NORMAL_FONT, 0, 50, 50, "テスト表¥示" );
}
```

留意事項

○ 文字化けについて

ほとんどの漢字は、2 重引用符で囲んだ漢字列の形でそのまま表示できます。
ただし、シフト JIS コードの LSByte が 0x5C(ASCII コードのバックスラッシュ)である文字については、C コンパイラがこのコードを棄却し、MSByte 部だけのコードとしてしまいます。
従って、その文字の続きに入っている文字コードの MSByte が、1 つ前の文字の LSByte となってしまいメチャクチャな表示となります。
十分に心掛けておいてください。
シフト JIS コードのうち MSByte が 0x5C となるコードはなく、また LSByte が 0x5C となるものも少ないので、漢字表示実施の場合には、ほとんどがこのシフト JIS コードにて表示できます。
LSByte が 0x5C となる文字については、表示不可文字に続けて “¥” マークを付加してください。

☆ LSByte が 0x5C となるシフト JIS コードの主なもの

ソ　欺　圭　構　蚕　十　申　貼　能　表　暴　予

Lib_display_comment

機　能

枠付きの文字列の表示

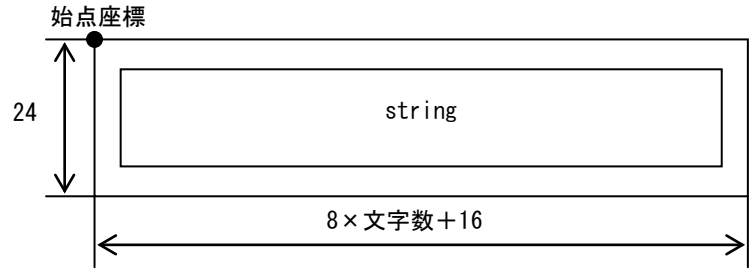
形　式

```
#include "f_graph.h"
void Lib_display_comment( int xpos, int ypos, char *string );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字プレーンに枠と文字列(ANKと旧JIS第1水準のシフトJIS)を任意の位置に表示します。
表示位置は枠左上隅を合わせます。
枠の大きさは



① ***xpos*** はX表示座標です。

② ***ypos*** はY表示座標です。

③ ****string*** は文字列へのポインタです。

戻り値

ありません。

例

```
#include "f_graph.h"

void disp_test()
{
    Lib_display_comment( 10, 10, "Test On Display" );
}
```

留意事項

○ Lib_kanjishift の留意事項を参照して下さい。

Lib_chrdisp

機　能

文字列描画

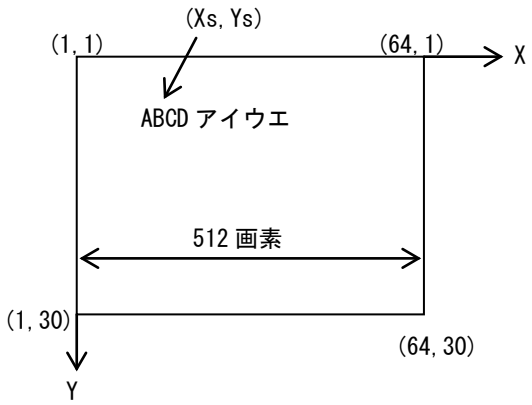
形　式

```
#include "f_graph.h"
int Lib_chrdisp( int xpos, int ypos, char *strings );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字列(ANKと旧JIS第1水準のシフトJIS)を文字フレームバッファ規定位置に表示します。



フォントサイズは
　☆ANK‥‥‥

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

文字列「ank　12Abab」を表示します。

```
#include "f_graph.h"

void disp_test()
{
    Lib_chrdisp( 10, 20, "ank 12ABab" );
}
```

留意事項

○ Lib_kanjishift の留意事項を参照して下さい。

Lib_invdisp

機　能

文字列反転表示

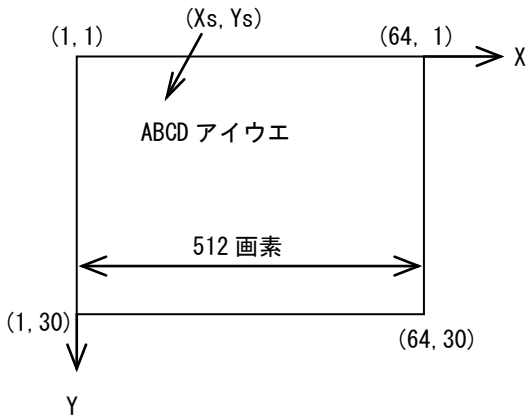
形　式

```
#include "f_graph.h"
int Lib_invdisp( int xpos, int ypos, char *strings );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字列(ANK と旧 JIS 第 1 水準のシフト JIS)を文字フレームバッファ規定位置に反転表示します。



フォントサイズは

☆ANK ・・・

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

文字列「ank　12Abab」を表示します。

```
#include "f_graph.h"

void disp_test()
{
    Lib_invdisp( 10, 20, "ank 12ABab" );
}
```

留意事項

○ Lib_kanjishift の留意事項を参照して下さい。

Lib_dispval

機　能

3 ケタ以内数値表示

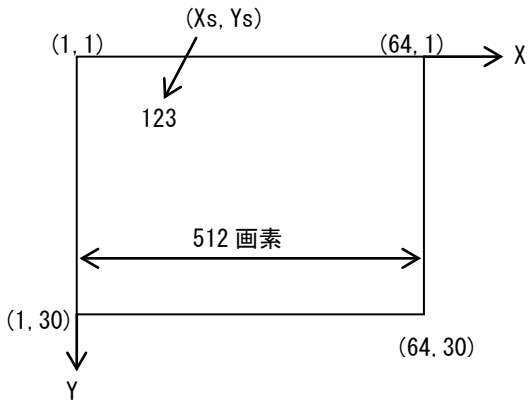
形　式

```
#include "f_graph.h"
int Lib_dispval( int xpos, int ypos, int value );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字フレームバッファに 3 ケタ以内の数値を規定位置に表示します。(3 ケタ固定)
但し、負の表示はーをつけた 3 ケタの数字列で表示されます。
〈例〉　3→003　　ー3→ー003



フォントサイズは　8(横)×16(縦)です。

① ***xpos*** は表示カラム位置(1～64)です。

② ***ypos*** は表示行位置(1～30)です。

③ ***value*** はー999 から 999 までの数値です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

数値「256」を表示します。

```
#include "f_graph.h"

void disp_test()
{
    int value;

    value = 256;
    Lib_dispval( 10, 20, value );
}
```

留意事項

ありません。

Lib_ddeciml

機　能

倍精度整数値表示

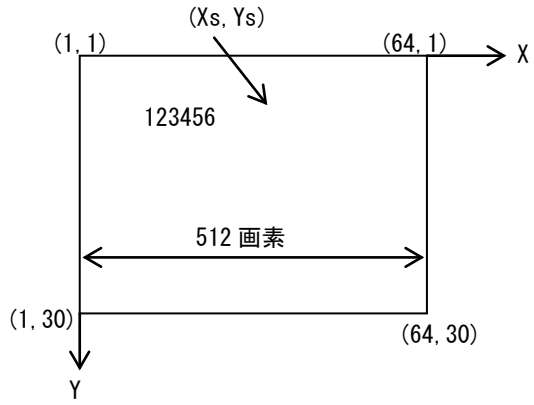
形　式

```
#include "f_graph.h"
int Lib_ddeciml( int xpos, int ypos, int value );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字フレームバッファに 0 から 999999 までの 6 ケタ整数値を規定位置に表示します。
但し、正の数のみを表示します。



フォントサイズは 8(横) × 16(縦) です。

① ***xpos*** は表示カラム位置(6～64)です。

② ***ypos*** は表示行位置(1～30)です。

③ ***value*** は 0 から 999999 までの数値です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

数値「123456」を表示します。

```
#include "f_graph.h"

void disp_test()
{
    int value;

    value = 123456;
    Lib_ddeciml( 10, 20, value );
}
```

留意事項

ありません。

Lib_pset

機　能

ドット描画

形　式

```
#include "f_graph.h"
int Lib_pset( int xpos, int ypos, int mode );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファにドットを描画または消去します。

① ***xpos*** は X 表示座標です。

② ***ypos*** は Y 表示座標です。

③ ***mode*** は作画モードです。

値	定　数	意　味
0	GRAPH_ERAS	ドットを消去します。
1	GRAPH_DRAW	ドットを表示します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

(25, 240)-(225, 240)の線を作画します。

```
#include "f_graph.h"
#define  Y_POS      240
#define  XS_POS     25
#define  XE_POS     225

void line()
{
    int x;

    for( x = XS_POS; x <= XE_POS; x++ )
        Lib_pset( x, Y_POS, GRAPH_DRAW );
}
```

留意事項

○ 位置が範囲外の場合は描画しません。

○ カーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。

Lib_line

機　能

直線描画(作画パタン指定)

形　式

```
#include "f_graph.h"
int Lib_line( int xs, int ys, int xe, int ye, int pattern );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファに指定作画パタンの直線を描画します。

① ***xs*** はＸ始端位置です。

② ***ys*** はＹ始端位置です。

③ ***xe*** はＸ終端位置です。

④ ***ye*** はＹ終端位置です。

⑤ ***pattern*** は作画パタンです。

戻り値

値	定　　数	意　　味
0xffffffff	SOLID_LINE	実線を作画します。
0x55555555	DOTTED_LINE	点線を作画します。
0x72727272	CENTER_LINE	一点鎖線を作画します。
0xf0f0f0f0	DASHED_LINE	破線を作画します。
0x00000000	BLACK_LINE	線を消去します。

上記以外は任意の値を指定してください。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

（25, 40）－（225, 300）の破線を作画します。

```
#include "f_graph.h"
#define  XS_POS      25
#define  XE_POS      225
#define  YS_POS      40
#define  YE_POS      300

void line()
{
    Lib_line( XS_POS, YS_POS, XE_POS, YE_POS, DASHED_LINE );
}
```

留意事項

○ 位置が範囲外の場合は描画しません。

○ カーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。

Lib_drawline

機　能

直線描画

形　式

```
#include "f_graph.h"
int Lib_drawline( int xs, int ys, int xe, int ye );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファに直線を描画します。

① ***xs*** はＸ始端位置です。

② ***ys*** はＹ始端位置です。

③ ***x*e** はＸ終端位置です。

④ ***ye*** はＹ終端位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

(25, 40)-(225, 300)の線を作画します。

```
#include "f_graph.h"
#define  XS_POS     25
#define  XE_POS     225
#define  YS_POS     40
#define  YE_POS     300

void line()
{
    Lib_drawline( XS_POS, YS_POS, XE_POS, YE_POS );
}
```

留意事項

○ 位置が範囲外の場合は描画しません。

○ カーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。

Lib_erasline

機　能

直線消去

形　式

```
#include "f_graph.h"
int Lib_erasline( int xs, int ys, int xe, int ye );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファ上の直線を消去します。

① *xs* は X 始端位置です。

② *ys* は Y 始端位置です。

③ *xe* は X 終端位置です。

④ *ye* は Y 終端位置です。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

(25, 40)-(225, 300)の線を消去します。

```
#include "f_graph.h"
#define  XS_POS     25
#define  XE_POS     225
#define  YS_POS     40
#define  YE_POS     300

void line()
{
    Lib_erasline( XS_POS, YS_POS, XE_POS, YE_POS );
}
```

留意事項

○ 位置が範囲外の場合は描画しません。

○ 901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。
902,903,904 ではカーソルの表示、非表示の影響は受けません。

Lib_box

機　能

ボックス線描画(作画パタン指定)

形　式

```
#include "f_graph.h"
int Lib_box( int xs, int ys, int xe, int ye, int pattern );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファに指定作画パタンのボックス線を描画します。

① ***xs*** はＸ始端位置です。

② ***ys*** はＹ始端位置です。

③ ***xe*** はＸ終端位置です。

④ ***ye*** はＹ終端位置です。

⑤ ***pattern*** は作画パタンです。

値	定　数	意　味
0xffffffff	SOLID_LINE	実線でボックス作画します。
0x55555555	DOTTED_LINE	点線でボックス作画します。
0x72727272	CENTER_LINE	一点鎖線でボックス作画します。
0xf0f0f0f0	DASHED_LINE	破線でボックス作画します。
0x00000000	BLACK_LINE	線を消去します。

上記以外は任意の値を指定してください。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

(25, 40)-(225, 300)のボックス線を破線で描画します。

```
#include "f_graph.h"
#define  XS_POS     25
#define  XE_POS     225
#define  YS_POS     40
#define  YE_POS     300

void box()
{
    Lib_box( XS_POS, YS_POS, XE_POS, YE_POS, DASHED_LINE );
}
```

留意事項

○ 位置が範囲外の場合は描画しません。

○ 901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。
902, 903, 904 ではカーソルの表示、非表示の影響は受けません。

Lib_drawcircle

機　能

円描画

形　式

```
#include "f_graph.h"
int Lib_drawcircle( int xo, int yo, int r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファに円を描画します。

① ***xo*** は中心 X 位置です。

② ***yo*** は中心 Y 位置です。

③ ***r*** は半径です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

中心(256, 240)の半径 128 の円を作画します。

```
#include "f_graph.h"
#define  X_POS      256
#define  Y_POS      240
#define  RADIUS     128

void circle()
{
    Lib_drawcircle( X_POS, Y_POS, RADIUS );
}
```

留意事項

○ 位置が範囲外の場合は描画しません。

○ 901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。
902, 903, 904 ではカーソルの表示、非表示の影響は受けません。

Lib_erascircle

機　能

円消去

形　式

```
#include "f_graph.h"
int Lib_erascircle( int xo, int yo, int r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファの円を消去します。

① ***xo*** は中心 X 位置です。

② ***yo*** は中心 Y 位置です。

③ ***r*** は半径です。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

中心（256, 240）の半径 128 の円を消去します。

```
#include "f_graph.h"
#define  X_POS      256
#define  Y_POS      240
#define  RADIUS     128

void circle()
{
    Lib_erascircle( X_POS, Y_POS, RADIUS );
}
```

留意事項

○ 位置が範囲外の場合は消去しません。

○ 901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。
902,903,904 ではカーソルの表示、非表示の影響は受けません。

Lib_circle

機　能

楕円描画

形　式

```
#include "f_graph.h"
int Lib_circle( int xo, int yo, int rad_long,
                int rad_short, int sin_value, int cos_value, int mode );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファに楕円を描画または消去します。

① ***xo*** は中心 X 位置です。

② ***yo*** は中心 Y 位置です。

③ ***rad_long*** は主軸半径(0～1023)です。

④ ***rad_short*** は副軸半径(0～1023)です。

⑤ ***sin_value*** は楕円の傾斜を表す正弦(sin)の 1024 倍値です。

⑥ ***cos_value*** は楕円の傾斜を表す余弦(cos)の 1024 倍値です。

⑦ ***mode*** は作画モードです。

値	定　数	意　味
0	GRAPH_ERAS	楕円を消去します。
1	GRAPH_DRAW	楕円を作画します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

中心(256, 240)の半径 128 の円と傾斜角 45 度の楕円を作画します。

```
#include <math.h>
#include "f_graph.h"
#define  X_POS      256
#define  Y_POS      240
#define  RADIUS     128
#define  ANGLE      45
#define  RAD_S      100
#define  RAD_L      128

void circle()
{
    int    wsin;
    int    wcos;

    wsin = Lib_sinfunc(ANGLE*10);
    wcos = Lib_cosfunc(ANGLE*10);
    wsin = wsin*1024/32767;
    wcos = wcos*1024/32767;

    Lib_circle( X_POS, Y_POS, RADIUS, RADIUS, 0, 0, GRAPH_DRAW );
    Lib_circle( X_POS, Y_POS, RAD_L, RAD_S, wsin, wcos, GRAPH_DRAW );
}
```

留意事項

○ 位置が範囲外の場合は描画/消去しません。

○ 901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。
902,903,904 ではカーソルの表示、非表示の影響は受けません。

Lib_box_cls

機　能

矩形塗りつぶし(クリア)

形　式

```
#include "f_graph.h"
int Lib_box_cls( int xs, int ys, int xe, int ye, int plane, int mode );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フレームバッファの各面を指定矩形範囲で黒または白で塗りつぶします。

① ***xs*** は X 始端位置です。

② ***ys*** は Y 始端位置です。

③ ***xe*** は X 終端位置です。

④ ***ye*** は Y 終端位置です。

⑤ ***plane*** はフレームバッファです。
(該当するメモリのビットをオンにしたビットパタンで指定します)

値	定　数	意　味
0x01	BIN_PLANE	2 値用フレームバッファを指定します。
0x20	GRAY_PLANE	濃淡用フレームバッファを指定します。
0x40	LINE_PLANE	線画用フレームバッファを指定します。
0x80	CHAR_PLANE	文字用フレームバッファを指定します。

⑥ ***mode*** は描画モードです。

値	定　数	意　味
0	BLACK_COLOR	黒で塗りつぶします。
0 以外	WHITE_COLOR	白で塗りつぶします。

濃淡の場合は濃度値です。(例えば、2 値用バッファを白で濃淡用のバッファの濃度値を 128 で塗りつぶしたい場合は、mode を 128 にします。)

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

線画、文字フレームバッファの (25, 40) -(225,300) 範囲を白で塗りつぶします。

```
#include "f_graph.h"
#define  XS_POS     25
#define  XE_POS     225
#define  YS_POS     40
#define  YE_POS     300

void box()
{
    Lib_box_cls (XS_POS, YS_POS, XE_POS, YE_POS,
                 (LINE_PLANE|CHAR_PLANE), WHITE_COLOR );
}
```

留意事項

○ 線画用フレームバッファを塗り潰す場合。

901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。

描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。

902, 903, 904 ではカーソルの表示、非表示の影響は受けません。

Lib_cls

機　能

フレームバッファのクリア

形　式

```
#include "f_graph.h"
int Lib_cls( int plane, int mode );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フレームバッファの各面を黒または白で塗りつぶします。

① ***plane*** はフレームバッファです。
（該当するメモリのビットをオンにしたビットパタンで指定します）

値	定　　数	意　　味
0x01	BIN_PLANE	2 値用フレームバッファを指定します。
0x20	GRAY_PLANE	濃淡用フレームバッファを指定します。
0x40	LINE_PLANE	線画用フレームバッファを指定します。
0x80	CHAR_PLANE	文字用フレームバッファを指定します。

② ***mode*** は描画モードです。

値	定　　数	意　　味
0	BLACK_COLOR	黒で塗りつぶします。
0 以外	WHITE_COLOR	白で塗りつぶします。

濃淡の場合は濃度値です。(例えば、2 値用バッファを白で濃淡用のバッファの濃度値を 128 で塗りつぶしたい場合は、mode を 128 にします。)

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

線画、文字フレームバッファを白で塗りつぶします。

```
#include "f_graph.h"

void box()
{
    Lib_cls ( (LINE_PLANE|CHAR_PLANE), WHITE_COLOR );
}
```

留意事項

○ 線画用フレームバッファをクリアする場合。

901 ではカーソルが表示されている場合、カーソル部分を退避しているために描画が遅くなります。
描画を高速に行ないたい場合はカーソルを消去してから描画し、描画終了後カーソルを復帰してください。
902,903,904 ではカーソルの表示、非表示の影響は受けません。

Lib_get

機　能

グラフィックデータ退避

形　式

```
#include "f_graph.h"
int Lib_get( int xs, int ys, int xe, int ye, int plane, char *buff );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フレームバッファ上のグラフィック･パタンを配列変数に読み込みます。

① ***xs*** は X 始端位置です。

② ***ys*** は Y 始端位置です。

③ ***xe*** は X 終端位置です。

④ ***ye*** は Y 終端位置です。

⑤ ***plane*** はフレームバッファです。
（該当するメモリのビットをオンにしたビットパタンで指定します）

値	定　数	意　味
0x01	BIN_PLANE	2 値用フレームバッファを読み込みます。
0x40	LINE_PLANE	線画用フレームバッファを読み込みます。
0x80	CHAR_PLANE	文字用フレームバッファを読み込みます。

⑥ ****buff*** は読み込みバッファへのポインタです。
バッファ容量は、下記の式で算出されるバイト数分必要とします。

（90X をご使用の場合）

$$\left(\left(Y_e - Y_s + 1\right) \times \left(\frac{X_e - X_s + 1}{8} + 1\right)\right) + 20$$

（FVL/LNX をご使用の場合）

$$\left(X_e - X_s + 1\right) \times \left(Y_e - Y_s + 1\right) \times 3 + 20$$

ただし、$Ye \geqq Ys$,　　$Xe \geqq Xs$ の場合

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

線画フレームバッファを全面退避します。

```
#include "f_graph.h"
#include "f_stdlib.h"

char *Save_Buff;

void shelter_plane()
{
    int size;
    int x_size;
    int y_size;

    x_size = Lib_get_fx_size();
    y_size = Lib_get_fy_size();
    size = y_size * (x_size / 8 + 1) + 20;
    if ( NULL != (Save_Buff = Lib_mlalloc( size )))
        Lib_get ( 0, 0, x_size - 1, y_size - 1, LINE_PLANE, Save_Buff );
}
```

留意事項

○ 退避するサイズよりバッファサイズが小さい場合、メモリ内容を破壊してしまいます。
バッファサイズには十分注意してください。

FVL/LNX

○ 本関数はバッファサイズの計算を間違えるとメモリを破壊してしまいます。
なるべく、“Lib_shelter_plane()”などの関数で置き換えることを検討してください。

Lib_put

機　能

グラフィックデータ復帰

形　式

```
#include "f_graph.h"
int Lib_put( char *buff );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_get ライブラリで読み込んだグラフィック･パタンをフレームバッファへ書き戻します。

① ***buff*** は退避しておいたグラフィック･パタンが格納されている配列変数へのポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

線画フレームバッファを全面退避し、処理後、復元します。

```
#include "f_graph.h"
#include "f_stdlib.h"

char *Save_Buff;

void shelter_plane(void);
void recover_plane(void);
void main()
{
    shelter_plane();
    /* 処理 */
    recover_plane();
}

void recover_plane( )         /*画面の復元*/
{
    Lib_put ( Save_Buff );
    Lib_lfree( Save_Buff );           /*バッファの解放*/
}

void shelter_plane()    /*線画画面の退避*/
{
    int size;
    int x_size;
    int y_size;

    x_size = Lib_get_fx_size();              /*X フレームバッファサイズの参照*/
    y_size = Lib_get_fy_size();              /*Y フレームバッファサイズの参照*/
    size = y_size * (x_size / 8 + 1) + 20;

    if ( NULL != (Save_Buff = Lib_mlalloc( size ))   /*バッファの確保*/
        Lib_get ( 0, 0, x_size, y_size, LINE_PLANE, Save_Buff );
}
```

留意事項

ありません。

Lib_shelter_plane

機　能

フレーム(グラフィック)･バッファの退避

形　式

```
#include "f_graph.h"
int Lib_shelter_plane( int xs, int ys, int xe, int ye, int plane );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定のフレーム(グラフィック)･バッファを指定範囲分退避します。
内部ではメモリをアロケーション後、関数 Lib_get により退避し、指定範囲をクリアしています。

① ***xs*** は退避面の X 位置始点です。

② ***ys*** は退避面の Y 位置始点です。

③ ***xe*** は退避面の X 位置終点です。

④ ***ye*** は退避面の Y 位置終点です。

⑤ ***plane*** はフレーム(グラフィック)･バッファです。
(該当するメモリのビットをオンにしたビットパタンで指定します)

値	定　数	意　味
0x01	BIN_PLANE	2 値用カレントメモリを指定します。
0x40	LINE_PLANE	線画用フレームバッファを指定します。
0x80	CHAR_PLANE	文字用フレームバッファを指定します。

ヒープ領域が確保できれば 50 枚分を退避する事ができます。

戻り値

処理結果

値	定　数	意　味
0<	なし	退避 ID です。
-1	ERROR_RETURN	異常終了です。

例

文字プレーンを退避し、濃淡メモリを表示します。

```
#include "f_graph.h"
#include "f_video.h"
#include "f_gui.h"

void display()
{
    int id;

    id = Lib_shelter_plane( 0, 0, (Lib_get_fx_size() - 1),
                                  (Lib_get_fy_size() - 1), CHAR_PLANE );

    Lib_xvideo_transmit( CURRENT_MEMORY, GRAY_PLANE );
    Lib_display_keyinput( 240, 230,"確認");

    Lib_recover_plane( id );
}
```

留意事項

○ プレーンは重複して指定できません。

Lib_recover_plane

機　能

フレーム(グラフィック)･バッファの復帰

形　式

```
#include "f_graph.h"
void Lib_recover_plane( int id );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_shelter_plane で取得した退避 ID によりグラフィック･バッファの復帰を行います。

① ***id*** は Lib_shelter_planen で取得した退避 ID です。

戻り値

ありません。

例

Lib_shelter_plane を参照してください。

留意事項

ありません。

Lib_view

機　能

矩形表示ウィンドウの生成/消滅

形　式

```
#include "f_graph.h"
int Lib_view( int xs, int ys, int xe, int ye, int mode );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

矩形表示ウィンドウの生成または消滅をおこないます。
Lib_window との相違は Lib_window は処理と表示範囲を設定しているのに対し、
Lib_view は表示のみを設定している点です。
ウィンドウはオーバーラップするため、重畳した場合、全領域をクリアしてから、必要なウィンドウ領域を再セットしなければなりません。

① ***xs*** は X 始端位置です。

② ***ys*** は Y 始端位置です。

③ ***xe*** は X 終端位置です。

④ ***ye*** は Y 終端位置です。

⑤ ***mode*** は作画モードです。

値	定　数	意　味
0	GRAPH_ERAS	矩形内を未表示にします。
1	GRAPH_DRAW	矩形内を表示します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

(25, 40) - (225,300) と (125,240) - (425,400)の範囲を表示可能にします。

```
#include "f_graph.h"

void set_view()
{
    Lib_view (   0,   0, 511, 479, GRAPH_ERAS );
    Lib_view (  25,  40, 225, 300, GRAPH_DRAW );
    Lib_view ( 125,240, 425, 400, GRAPH_DRAW );
}
```

留意事項

○ 本ライブラリは GUI ライブラリ中のパッドとは直接関係ありません。

Lib_paint

機　能

ペイント

形　式

```
#include "f_graph.h"
int Lib_paint( int xpos, int ypos, int color, int ref_color, int plane );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

指定された位置を始点として、境界色で囲まれた領域を、指定された領域色で塗りつぶします。

① ***xpos*** は、始点 X 位置です。

② ***ypos*** は、始点 Y 位置です。

③ ***color*** は、領域色です。

値	定　数	意　味
0	BLACK_COLOR	領域色に黒を指定します。
1	WHITE_COLOR	領域色に白を指定します。

④ ***ref_color*** は、境界色です。

値	定　数	意　味
0	BLACK_COLOR	境界色に黒を指定します。
1	WHITE_COLOR	境界色に白を指定します。

⑤ ***plane*** は、塗り込みプレーンです。

値	定　数	意　味
0x01	BIN_PLANE	2 値用カレントメモリを指定します。
0x40	LINE_PLANE	線画用フレームバッファを指定します。
0x80	CHAR_PLANE	文字用フレームバッファを指定します。

戻り値

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

点列のプロット描画を行い、その囲まれた領域を塗りつぶします。

```
#include "f_graph.h"
#define   PLOT_NO 25
const struct COORDINATES_PAIR xx[PLOT_NO] = {256,  20,
                                             226, 128,
                                             256, 156,
                                             366,  50,
                                             446, 130,
                                             338, 158,
                                             382, 198,
                                             476, 240,
                                             446, 350,
                                             366, 270,
                                             328, 282,
                                             366, 430,
                                             256, 460,
                                             286, 352,
                                             256, 324,
                                             146, 430,
                                              66, 350,
                                             174, 322,
                                             184, 282,
                                              34, 240,
                                              64, 130,
                                             144, 208,
                                             186, 198,
                                             144, 50,
                                             256, 20
                                             };
void paint_test()
{
    Lib_plot( PLOT_NO,(struct COORDINATES_PAIR *) xx, GRAPH_DRAW, LINE_PLOT );
    Lib_paint( 256, 240, WHITE_COLOR, WHITE_COLOR,LINE_PLANE  );
}
```

留意事項

○ プレーンで 2 値を指定した場合はカレントメモリを処理対象としています。
したがって処理結果はフレームバッファに転送しないと表示されません。

○ 参照プレーンは常に線画用フレームバッファです。

○ 塗り込みプレーンは重複して指定できません。

Lib_paint2

機　能

ペイント２

形　式

```
#include "f_graph.h"
int Lib_paint2(  int xpos, int ypos, int color,
                                int ref_color, int ref_plane, int plane );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

参照プレーンの指定された位置を中心点として、境界色で囲まれた領域を、指定された領域色で指定塗り込みプレーンを塗りつぶします。

① ***xpos*** は始点Ｘ位置です。

② ***ypos*** は始点Ｙ位置です。

③ ***color*** は領域色（塗りたい色）です。

値	定　　数	意　　味
0	BLACK_COLOR	領域色に黒を指定します。
1	WHITE_COLOR	領域色に白を指定します。

④ ***ref_color*** は境界色です。

値	定　　数	意　　味
0	BLACK_COLOR	境界色に黒を指定します。
1	WHITE_COLOR	境界色に白を指定します。

⑤ ***ref_plane*** は参照プレーンです。

値	定　　数	意　　味
0x01	BIN_PLANE	2 値用フレームバッファを指定します。
0x40	LINE_PLANE	線画用フレームバッファを指定します。
0x80	CHAR_PLANE	文字用フレームバッファを指定します。

⑥ ***plane*** は指定塗り込みプレーンです。

値	定　　数	意　　味
0x01	BIN_PLANE	2 値用カレントメモリを指定します。
0x40	LINE_PLANE	線画用フレームバッファを指定します。
0x80	CHAR_PLANE	文字用フレームバッファを指定します。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

線画用プレーンに矩形を描画し、その内側部分の 2 値画像メモリをペイントします。

```
#include "f_graph.h"

void paint_test()
{
    Lib_box( 10, 10, 400, 400, SOLID_LINE );
    Lib_paint2(  100, 100, WHITE_COLOR, WHITE_COLOR, LINE_PLANE, BIN_PLANE );
}
```

留意事項

○ プレーンは重複して指定できません。

Lib_plot

機　能

点列プロット

形　式

```
#include "f_graph.h"
int Lib_plot( int no, struct COORDINATES_PAIR  *pair_buf, int mode, int type );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

線画用フレームバッファに点列のプロットの描画または削除をおこないます。

① ***no*** は、点列の長さ(座標ペアの要素数)です。

② ****pair_buf*** は、点列座標ペア配列へのポインタです。

```
struct COORDINATES_PAIR    {
                               short int x
                               short int y
                           };
```

③ ***mode*** は、書き込みフラグです。

値	定　　数	意　　味
0	GRAPH_ERAS	プロット列を削除します。
1	GRAPH_DRAW	プロット列を描画します。

④ ***type*** は、プロットタイプです。

値	定　　数	意　　味
0	DOT_PLOT	点列をドット･プロットします。
1	LINE_PLOT	点列をライン･プロットします。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

Lib_paint の例を参照してください。

留意事項

ありません。

Lib_xbline

機　能

2 値画像メモリ直線描画

形　式

```
#include "f_graph.h"
int Lib_xbline( int memory_no, int color, int xs, int ys, int xe, int ye );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

2 値画像メモリに直線を描画します。

① ***memory_no*** は 2 値画像メモリ番号. です。

② ***color*** は描画する色です。

値	定　　数	意　　味
0	BLACK_COLOR	黒線を描画します。
1	WHITE_COLOR	白線を描画します。

③ ***xs*** は X 始点位置です。

④ ***ys*** は Y 始点位置です。

⑤ ***xe*** は X 終端位置です。

⑥ ***ye*** は Y 終端位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント 2 値画像メモリに(0, 0)-(511, 479)の直線を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void xbline( void )
{
    Lib_display_control( BIN_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( BIN_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_xbline( CURRENT_MEMORY, BLACK_COLOR, 0, 0, 511, 479 );

    Lib_video_transmit( BIN_PLANE );
}
```

留意事項

○ 始点、終点位置は 2 値画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は 2 値画像メモリの範囲となります。

Lib_zbline

機　能

2 値画像メモリ逆色直線描画

形　式

#include "f_graph.h"
int Lib_zbline(int ***memory_no***, int ***xs***, int ***ys***, int ***xe***, int ***ye***);

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

2 値画像メモリに、メモリの内容を反転(0 ならば 1、1 ならば 0)下直線を描画します。始点、終点位置を変えずに 2 回続けて呼び出すと、元のメモリの内容に戻ります。

① ***memory_no*** は 2 値画像メモリ番号. です。

② ***xs*** は X 始点位置です。

③ ***ys*** は Y 始点位置です。

④ ***xe*** は X 終端位置です。

⑤ ***ye*** は Y 終端位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント 2 値画像メモリに(0, 0)-(511, 479)の逆色直線を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void zbline( void )
{
    Lib_display_control( BIN_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( BIN_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_zbline( CURRENT_MEMORY, 0, 0, 511, 479 );

    Lib_video_transmit( BIN_PLANE );
}
```

留意事項

○ 始点、終点位置は 2 値画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は 2 値画像メモリの範囲となります。

Lib_xbcircle

機　能

2 値画像メモリ円描画

形　式

```
#include "f_graph.h"
int Lib_xbcircle( int memory_no, int color, int xo, int yo, int r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

2 値画像メモリに円を描画します。

① ***memory_no*** は 2 値画像メモリ番号. です。

② ***color*** は描画する色です。

値	定　　数	意　　味
0	BLACK_COLOR	黒円を描画します。
1	WHITE_COLOR	白円を描画します。

③ ***xo*** は中心 X 位置です。

④ ***yo*** は中心 Y 位置です。

⑤ ***r*** は半径です。
指定範囲は 1 から 1023 です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント 2 値画像メモリに中心(250, 200)、半径 100 の円を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void xbcircle( void )
{
    Lib_display_control( BIN_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( BIN_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_xbcircle( CURRENT_MEMORY, BLACK_COLOR, 250, 200, 100 );

    Lib_video_transmit( BIN_PLANE );
}
```

留意事項

○ 中心位置は 2 値画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は 2 値画像メモリの範囲となります。

Lib_zbcircle

機　能

2 値画像メモリ逆色円描画

形　式

```
#include "f_graph.h"
int Lib_zbcircle( int memory_no, int xo, int yo, int r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

2 値画像メモリに、メモリの内容を反転(0 ならば 1、1 ならば 0)下円を描画します。
始点、終点位置を変えずに 2 回続けて呼び出すと、元のメモリの内容に戻ります。

① ***memory_no*** は 2 値画像メモリ番号.です。

② ***xo*** は中心 X 位置です。

③ ***yo*** は中心 Y 位置です。

④ ***r*** は半径です。
指定範囲は 1 から 1023 です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント 2 値画像メモリに中心(250, 200)、半径 100 の逆色円を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void zbcircle( void )
{
    Lib_display_control( BIN_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( BIN_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_zbcircle( CURRENT_MEMORY, 250, 200, 100 );

    Lib_video_transmit( BIN_PLANE );
}
```

留意事項

○ 中心位置は 2 値画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は 2 値画像メモリの範囲となります。

Lib_xgline

機　能

濃淡画像メモリ直線描画

形　式

```
#include "f_graph.h"
int Lib_xgline( int memory_no, int color, int xs, int ys, int xe, int ye );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

メモリに直線を描画します。

① ***memory_no*** は濃淡画像メモリ番号.です。

② ***color*** は描画する濃度値です。
指定範囲は 0 から 255 です。

③ ***xs*** は X 始点位置です。

④ ***ys*** は Y 始点位置です。

⑤ ***xe*** は X 終端位置です。

⑥ ***ye*** は Y 終端位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに(0, 0)-(511, 479)、濃度値 255 の直線を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void xgline( void )
{
    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_xgline( CURRENT_MEMORY, 255, 0, 0, 511, 479 );

    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 始点、終点位置は濃淡画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は濃淡画像メモリの範囲となります。

Lib_zgline

機　能

濃淡画像メモリ逆色直線描画

形　式

```
#include "f_graph.h"
int Lib_zgline( int memory_no, int xs, int ys, int xe, int ye );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリに、メモリの内容を反転(0 ならば 1、1 ならば 0)した直線を描画します。
始点、終点位置を変えずに 2 回続けて呼び出すと、元のメモリの内容に戻ります。

① ***memory_no*** は濃淡画像メモリ番号.です。

② ***xs*** は X 始点位置です。

③ ***ys*** は Y 始点位置です。

④ ***xe*** は X 終端位置です。

⑤ ***ye*** は Y 終端位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに(0, 0)-(511, 479)の逆色直線を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void zgline( void )
{
    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_zgline( CURRENT_MEMORY, 0, 0, 511, 479 );

    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 始点、終点位置は濃淡画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は濃淡画像メモリの範囲となります。

Lib_xgcircle

機　能

濃淡画像メモリ円描画

形　式

```
#include "f_graph.h"
int Lib_xgcircle( int memory_no, int color, int xo, int yo, int r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

濃淡画像メモリに円を描画します。

① ***memory_no*** は濃淡画像メモリ番号.です。

② ***color*** は描画する濃度値です。
指定範囲は 0 から 255 です。

③ ***xo*** は中心 X 位置です。

④ ***yo*** は中心 Y 位置です。

⑤ ***r*** は半径です。
指定範囲は 1 から 1023 です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに中心(250, 200)、半径 100、濃度値 255 の円を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void xgcircle( void )
{
    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_xgcircle( CURRENT_MEMORY, 255, 250, 200, 100 );

    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 中心位置は濃淡画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は濃淡画像メモリの範囲となります。

Lib_zgcircle

機　能

濃淡画像メモリ逆色円描画

形　式

```
#include "f_graph.h"
int Lib_zgcircle( int memory_no, int xo, int yo, int r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリに、メモリの内容を反転(0 ならば 1、1 ならば 0)下円を描画します。
始点、終点位置を変えずに 2 回続けて呼び出すと、元のメモリの内容に戻ります。

① ***memory_no*** は濃淡画像メモリ番号.です。

② ***xo*** は中心 X 位置です。

③ ***yo*** は中心 Y 位置です。

④ ***r*** は半径です。
指定範囲は 1 から 1023 です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに中心(250,200)、半径 100 の逆色円を描画します。

```
#include "f_video.h"
#include "f_graph.h"

void zgcircle( void )
{
    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );

    Lib_zgcircle( CURRENT_MEMORY, 250, 200, 100 );

    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 中心位置は濃淡画像メモリの範囲(描画可能範囲)を越えて指定できますが、その絶対値が大きい場合、描画がうまく行かない場合があります。
また、実際の描画は濃淡画像メモリの範囲となります。

Lib_arc_box

機　能

円弧ボックス線描画(作画パタン指定)

形　式

```
#include "f_graph.h"
int Lib_arc_box( int xc, int yc, int rs, double qs,
                                 int re, double qe, int pattern );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

線画フレームバッファに指定作画パタンの円弧ボックス線を描画します。

① ***xc*** は円弧中心 X 座標を指定します。

② ***yc*** は円弧中心 Y 座標を指定します。

③ ***rs*** は扇型エリアの領域の始点半径を画素単位で指定します。
(始点半径 > 終点半径としてください)

④ ***qs*** は扇型エリアの領域の始点角度です。
時計の 3 時方向を 0.0 とした時計回りで指定します。(0.0. . 360.0)

⑤ ***re*** は扇型エリアの領域の終点半径を画素単位で指定します。
(始点半径 > 終点半径としてください)

⑥ ***qe*** は扇型エリアの領域の終点角度です。
時計の 3 時方向を 0.0 とした時計回りで指定します。(0.0. . 360.0)

⑦ ***pattern*** は作画パタンです。

値	定　数	意　味
0xffffffff	SOLID_LINE	実線を作画します。
0x55555555	DOTTED_LINE	点線を作画します。
0x72727272	CENTER_LINE	一点鎖線を作画します。
0xf0f0f0f0	DASHED_LINE	破線を作画します。
0x00000000	BLACK_LINE	黒線を作画します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	パラメータエラーです。

例

円弧中心(255, 240)、始点半径 200、終点半径 100、始点角度 0.0、終点角度 270.0 で示される円弧ボックス線を実線で描画します。

```
#include "f_graph.h"
#define  XC             255
#define  YC             240
#define  RS             200
#define  QS             0.0
#define  RE             100
#define  QE             270.0

void arc_box()
{
    Lib_arc_box( XC, YC, RS, QS, RE, QE, SOLID_LINE );
}
```

留意事項

ありません。

Lib_vout_create_video_window

機　能

ビデオウィンドウの生成

形　式

```
#include "f_vout.h"
int Lib_vout_create_video_window( int xs, int ys, int dxsize,
                                  int dysize, char * title );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

新規にビデオウィンドウを生成します。

① ***xs*** はビデオウィンドウの横方向始点座標です。

② ***ys*** はビデオウィンドウの横方向始点座標です。

③ ***dxsize*** はビデオウィンドウの横方向画素数です。

④ ***dysize*** はビデオウィンドウの縦方向画素数です。

⑤ ***title*** はビデオウィンドウのタイトル文字列です。
NULL を指定するとデフォルト文字列“video window”が設定されます。

戻り値

処理結果

値	定　　数	意　　味
0 以上	ありません	ビデオウィンドウ番号です。
-1	ERROR_RETURN	異常終了しました。
-10002	ERR_ILLEGAL_PARAM	引数に異常があります。
-10003	ERR_NO_MEMORY	メモリ不足です。

例

新規にビデオウィンドウを確保して、2枚のビデオウィンドウに文字の描画を行ないます。

```
/* include */
#include "f_video.h"
#include "f_graph.h"
#include "f_gui.h"
#include "f_vout.h"

int main( void )
{
    int     iVideoNo0, iVideoNo1;

    Lib_stop_freerun();
    Lib_cls( GRAY_PLANE, BLACK_COLOR );

    iVideoNo0 = Lib_vout_get_video_no();
    Lib_vout_move_video_window( CURRENT_VIDEO_NO, 0, 128 );

    if( 0 <= ( iVideoNo1 = Lib_vout_create_video_window
                                            ( 500, 128, 512, 480, NULL ) ) )
    {
        Lib_chrdisp( 1, 1, "Video Window 0" );

        Lib_vout_set_video_no( iVideoNo1 );
        if( 0xffffff/* White */ == Lib_vout_get_color( CHAR_PLANE ) )
                    Lib_vout_set_color( CHAR_PLANE, 0xff0000/* Red */ );
        Lib_chrdisp( 1, 1, "Video Window 1" );

        Lib_display_message( 0, 0, "", "終了します" );
        Lib_vout_set_video_no( iVideoNo0 );
        Lib_vout_destroy_video_window( iVideoNo1 );
    }

    return( 0 );
}
```

留意事項

○ ビデオウィンドウはユーザプログラム起動時に1枚生成され、カレントビデオウィンドウに設定されます。

○ 指定された始点位置に表示できない場合は、画面中央に表示します。

Lib_vout_destroy_video_window

機　能

ビデオウィンドウの破棄

形　式

```
#include "f_vout.h"
int Lib_vout_destroy_video_window( int video_no );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

ビデオウィンドウを破棄します。

① ***video_no*** はビデオウィンドウ番号です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了です。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

ビデオウィンドウの生成(Lib_vout_create_video_window)を参照してください。

留意事項

○ カレントビデオウィンドウは破棄できません。

Lib_vout_get_video_no

機　能　カレントビデオウィンドウ番号の取得

形　式

```
#include "f_vout.h"
int Lib_vout_get_video_no( void );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説　グラフィックライブラリ、映像出力ライブラリなどの出力先ウィンドウのID番号(カレントビデオウィンドウ番号)を取得します。

戻り値　カレントビデオ番号

例　ビデオウィンドウの生成(Lib_vout_create_video_window)を参照してください。

留意事項　ありません。

Lib_vout_set_video_no

機　能

カレントビデオ番号の設定

形　式

```
#include "f_vout.h"
int Lib_vout_set_video_no( int video_no );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

グラフィックライブラリ、映像出力ライブラリなどの出力先ウィンドウのID番号(カレントビデオウィンドウ番号)を設定します。

① ***video_no*** はビデオウィンドウ番号です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了です。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

ビデオウィンドウの生成(Lib_vout_create_video_window)を参照してください。

留意事項

ありません。

Lib_vout_get_color

機　能

描画色の参照

形　式

```
#include "f_vout.h"
unsigned int Lib_vout_get_color( unsigned int plane );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

グラフィックライブラリの描画色を参照します。

① ***plane*** はビデオプレーン ID です。

値	定　数	意　味
0x00000040	LINE_PLANE	線画用フレームバッファを表示します。
0x00000080	CHAR_PLANE	文字用フレームバッファを表示します。

戻り値

24Bitカラー値

RGB各色8Bitで以下のように並んでいます。

	R	G	B

例

ビデオウィンドウの生成(Lib_vout_create_video_window)を参照してください。

留意事項

ありません。

Lib_vout_set_color

機　能

描画色の設定

形　式

```
#include "f_vout.h"
unsigned int Lib_vout_set_color( unsigned int plane, unsigned int color );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

グラフィックライブラリの描画色を設定します。

① ***plane*** はビデオプレーン ID です。

値	定　　数	意　　味
0x00000040	LINE_PLANE	線画用フレームバッファを表示します。
0x00000080	CHAR_PLANE	文字用フレームバッファを表示します。

② ***color*** は描画色です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了です。
-1	ERROR_RETURN	異常終了しました。

例

ビデオウィンドウの生成(Lib_vout_create_video_window)を参照してください。

留意事項

ありません。

Lib_vout_move_video_window

機　能

ビデオウィンドウの移動

形　式

```
#include "f_vout.h"
int Lib_vout_move_video_window( int video_no, int xs, int ys );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

指定されたビデオウィンドウ番号の始点位置を移動します。

① ***video_no*** はビデオウィンドウ番号です。

② ***xs*** はビデオウィンドウの横方向始点座標です。

③ ***ys*** はビデオウィンドウの横方向始点座標です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了です。
-1	ERROR_RETURN	異常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

ビデオウィンドウの生成(Lib_vout_create_video_window)を参照してください。

留意事項

ありません。

Lib_vout_get_geometry

機　能

ビデオウィンドウ情報の取得

形　式

```
#include "f_vout.h"
int Lib_vout_get_geometry( int video_no, int *sx, int *sy, int *dx, int *dy )
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

ビデオウィンドウ情報を取得します。

① *sx* はビデオウィンドウ右上表示位置です。(X 座標)

② *sy* はビデオウィンドウ右上表示位置です。(Y 座標)

③ *dx* はビデオウィンドウ横方向サイズです。

④ *dy* はビデオウィンドウ縦方向サイズです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了です。
-1	ERROR_RETURN	異常終了しました。

例

ビデオウィンドウの情報を取得して表示します。

```
#include "f_stdlib.h"
#include "f_graph.h"
#include "f_vout.h"

int disp_voutgeom( int video_no )
{
        int ret, sx, sy, dx, dy;
        char cbuf[256];

        if((ret = Lib_vout_get_geometry( video_no, &sx, &sy, &dx, &dy ))
          == NORMAL_RETURN)
        {
                Lib_sprintf( cbuf, "[%d] vout info: start(%d,%d) size(%d,%d)",
                             video_no, sx, sy, dx, dy );
                Lib_chrdisp( 1, 1, cbuf );
        }

        return ret;
}
```

留意事項

ありません。

Lib_vout_get_resolution

機　能

画面解像度の取得

形　式

```
#include "f_vout.h"
int Lib_vout_get_resolution( unsigned int *width, unsigned int *height, unsigned int *bit )
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

画面の解像度を取得します。

① ***width*** は画面の横の解像度です。

② ***height*** は画面の縦の解像度です。

③ ***bit*** は画面のビット深度です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了です。
-1	ERROR_RETURN	異常終了しました。

例

画面解像度を取得して表示します。

```
#include "f_stdlib.h"
#include "f_graph.h"
#include "f_vout.h"

int disp_voutgeom( int video_no )
{
        int ret, width, height, bit;
        char cbuf[256];

        if(NORMAL_RETURN==Lib_vout_get_resolution(&width,&height,&bit)
          == NORMAL_RETURN)
        {
                Lib_sprintf( cbuf, "[%dx%d:%d]",
                             Width, height, bit );
                Lib_chrdisp( 1, 1, cbuf );
        }

        return ret;
}
```

留意事項

ありません。

Lib_vout_sync_framebuffer

機　能

フレームバッファとローカルメモリの同期

形　式

```
#include "f_vout.h"
int Lib_vout_sync_framebuffer( int video_no, BOX_T area[], int area_num,
                                                    unsigned int plane );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

描画指示を発行し、ローカルメモリ上の描画データを画面に反映します。

① ***video_no*** はビデオウィンドウ番号です。

② ***area[]*** は描画エリア構造体の配列です。

BOX_T構造体は“f_type.h”で以下のように定義されています。

```
typedef struct
{
     PNT_T   st;     /* 開始点 */
     PNT_T   ed;     /* 終了点 */
} BOX_T;
```

③ ***area_num*** は描画エリア構造体の配列の個数です。

④ ***plane*** はビデオプレーン ID です。

値	定　数	意　味
0x00000001	BIN_PLANE	2 値用フレームバッファを表示します。
0x00000020	GRAY_PLANE	濃淡用フレームバッファを表示します。
0x00000040	LINE_PLANE	線画用フレームバッファを表示します。
0x00000080	CHAR_PLANE	文字用フレームバッファを表示します。
0x00000100	COLOR_PLANE	カラー用フレームバッファを表示します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了です。
-1	ERROR_RETURN	異常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

画面中央のY座標120～384の範囲に水平線を描画し、全て書き終わってから画面に反映します。

```
/* include */
#include "f_stdlib.h"
#include "f_video.h"
#include "f_graph.h"
#include "f_gui.h"
#include "f_vout.h"

int main( void )
{
    BOX_T      tArea;
    int iMode;
    int y;



    Lib_stop_freerun();
    Lib_cls( GRAY_PLANE, BLACK_COLOR );

    Lib_display_control( VOUT_NOT_SYNC|LINE_PLANE );
    tArea.st.x = tArea.st.y = 0;
    tArea.ed.x = Lib_get_dx_size()-1;
    tArea.ed.y = Lib_get_dy_size()-1;

    iMode = SOLID_LINE;
    for( ; ; )
    {
        for( y = 120; y <= 360; y++ )
                    Lib_line( 128, y, 384, y, iMode );
        Lib_vout_sync_framebuffer( CURRENT_VIDEO_NO, &tArea, 1, LINE_PLANE );

        if( SOLID_LINE == iMode )     iMode = BLACK_LINE;
        else                    iMode = SOLID_LINE;
        if( 0 != Lib_see_current_position_real( NULL, NULL ) )
        break;
    }

    return( 0 );
}
```

留意事項

○ 本ライブラリを使用する場合は、Lib_display_control()とあわせてご使用ください。

Lib_vout_set_video_window_title

機　能

ビデオウィンドウのタイトル変更

形　式

```
#include "f_vout.h"
int Lib_vout_set_video_window_title( int video_no, char *title );
```

901	902	903	904	FVL/LNX	高分解能
				○	○

解　説

ビデオウィンドウのタイトルを変更します。

① ***video_no*** はビデオウィンドウ番号です。

② ***title*** はビデオウィンドウのタイトル文字列です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

プログラム開始時に生成されるビデオウィンドウのタイトルを変更します。

```
void main( void )
{

        Lib_vout_set_video_window_title( Lib_vout_get_video_no(), “New Title” );
        /*****  メイン関数での処理を記述します  *****/

}
```

留意事項

ありません。

Lib_vout_draw_video_window

機　能

ビデオウィンドウの描画

形　式

```
#include "f_vout.h"
int Lib_vout_draw_video_window( int video_no );
```

901	902	903	904	FVL/LNX	高分解能
				○	○

解　説

指定されたビデオウィンドウを再描画します。

① ***video_no*** はビデオウィンドウ番号です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

カレントビデオウィンドウを消去し、何らかの処理をした後に再描画します。

```
void func( void )
{

        Lib_vout_erase_video_window( Lib_vout_get_video_no() );
        /*****  何らかの処理  *****/
        Lib_vout_draw_video_window( Lib_vout_get_video_no() );

}
```

留意事項

ありません。

Lib_vout_erase_video_window

機　能

ビデオウィンドウの消去

形　式

```
#include "f_vout.h"
int Lib_vout_erase_video_window( int video_no );
```

901	902	903	904	FVL/LNX	高分解能
				○	○

解　説

指定されたビデオウィンドウを消去します。

① ***video_no*** はビデオウィンドウ番号です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

Lib_vout_draw_video_window() の例をご覧ください。

留意事項

ありません。

Lib_vout_get_option

機　能

ビデオウィンドウプションの取得

形　式

```
#include "f_vout.h"
VWINATTR_T Lib_vout_get_option( void );
```

901	902	903	904	FVL/LNX	高分解能
				○	○

解　説

ビデオウィンドウ生成時のオプション設定を取得します。
構造体 VWINATTR_T は f_vout.h の内で次のように定義されています。

```
typedef struct
{
  unsigned int btTitleType    :1,     /*  0：タイトルなし(移動処理なし)
                                          1：タイトルあり(移動処理あり)  */
                              :31;
} VWINATTR_BIT;

typedef union
{
  unsigned int uiVwinAttr;
  VWINATTR_BIT tVwinAttrBit;
} VWINATTR_T;
```

戻り値

ビデオウィンドウプション構造体

例

Lib_vout_set_option() の例をご覧ください。

留意事項

ありません。

Lib_vout_set_option

機　能

ビデオウィンドウプションの変更

形　式

```
#include "f_vout.h"
int Lib_vout_set_option( VWINATTR_T vwin_attr );
```

901	902	903	904	FVL/LNX	高分解能
				○	○

解　説

ビデオウィンドウ生成時のオプション設定を変更します。

① ***vwin_attr*** はビデオウィンドウプションです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

タイトルなしのウィンドウを生成します。

```
void func( void )
{
  VWINATTR_T   tVwinAttr;

  tVwinAttr = Lib_vout_get_option();
  tVwinAttr.tVwinAttrBit.btTitleType = 0;     /* タイトルなし */
  Lib_vout_set_option( tVwinAttr );
  Lib_vout_create_video_window( 100, 100, 512, 480, "" );
}
```

留意事項

ありません。

3. メモリ領域割当ライブラリ

本ライブラリはヒープ領域から指定されたサイズのメモリを割当てるものです。

Lib_alloc_gray_memory

機　能

濃淡画像メモリの確保

形　式

```
#include "f_stdlib.h"
int Lib_alloc_gray_memory( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリ 1 面分(256K バイト)のメモリ用ファイルを確保します。

90X をご使用の場合

なお、確保されたメモリ領域はランダムファイルとして生成され、ファイルシステムを通してアロケーションを行ないます。
ファイル名は「GRAYMEM?. SYS」(?=メモリ番号.16 進数)と自動的につけられます。(確保できるのは 16 面目(GRAYMEMF. SYS)までです)

FVL/LNX をご使用の場合

90X では画像メモリをファイルシステム経由で確保していたため“STORAGE“ディレクトリにファイルを生成していましたが、FVL/LNX では純粋にメモリの確保のみを行ないます。また、画像メモリの確保枚数制限もありません。メモリの許す限り画像メモリを確保することが出来ます。

戻り値

確保したメモリ番号.を返します。

値	定　数	意　味
0 以上	ありません。	確保したメモリ番号.を意味します。
-1	ERROR_RETURN	異常終了しました。

例

メモリを新たに確保し、ビデオを入力します。

```
#include "f_stdlib.h"
#include "f_video.h"

void new_memory()
{
    int mem_no;

    if ( ERROR_RETURN != (mem_no = Lib_alloc_gray_memory( )) )
    {
        if ( NORMAL_RETURN == Lib_change_gray_memory ( mem_no ) )
        {
            Lib_input_video_control( GRAY_PLANE );

            Lib_freeze( TRANSMIT );
        }
        /* メモリの解放 */
        Lib_free_gray_memory( mem_no );
    }
}
```

留意事項　ありません。

FVL/LNX

○ FVL/LNX では 2 値画像メモリと濃淡画像メモリのメモリ番号が統合されています。
例えば、2 値、濃淡メモリをそれぞれ 2 枚ずつ確保したとすると
(90X での動作)
2 値画像メモリ：0,1
濃淡メモリ：0,1
(FVL/LNX での動作)
2 値画像メモリ：0,1
濃淡メモリ：2,3
となります。90X では 2 値画像メモリと濃淡メモリの ID 番号は重複しておりましたが、FVL/LNX では画像メモリの ID 番号は重複しません。
特に 90X からの移植をする際にはカレントメモリ番号が 0 とは限りませんので、注意して移植してください。

903, 904

○ 濃淡画像メモリを 32 面取ることが可能です。
ファイル名は 16 面以降「G000001?. SYS」(?=メモリ番号.16　進数)と自動的につけられます。

Lib_alloc_bin_memory

機　能

2 値画像メモリの確保

形　式

```
#include "f_stdlib.h"
int Lib_alloc_bin_memory( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

2 値画像メモリ 1 面分(32K バイト)のメモリ用ファイルを確保します。

90X をご使用の場合

なお、確保されたメモリ領域はランダムファイルとして生成され、ファイルシステムを通してアロケーションを行ないます。
ファイル名は「BIN__MEM?. SYS」(?＝メモリ番号.16 進数)と自動的につけられます。(確保できるのは 16 面目(BIN_MEMF.SYS)までです)

FVL/LNX をご使用の場合

90X では画像メモリをファイルシステム経由で確保していたため“STORAGE“ディレクトリにファイルを生成していましたが、FVL/LNX では純粋にメモリの確保のみを行ないます。また、画像メモリの確保枚数制限もありません。メモリの許す限り画像メモリを確保することが出来ます。

戻り値

確保したメモリ番号.を返します。

値	定　数	意　味
0 以上	ありません。	確保したメモリ番号.を意味します。
-1	ERROR_RETURN	異常終了しました。

例

メモリを新たに確保し、ビデオを入力します。

```
#include "f_stdlib.h"
#include "f_video.h"

void new_memory()
{
    int mem_no;

    if ( ERROR_RETURN != (mem_no = Lib_alloc_bin_memory( )) )
    {
        if ( NORMAL_RETURN == Lib_change_bin_memory ( mem_no ) )
        {
            Lib_input_video_control( BIN_PLANE );

            Lib_freeze( TRANSMIT );
        }
        /* メモリの解放 */
        Lib_free_bin_memory( mem_no );
    }
}
```

留意事項　ありません。

FVL/LNX

○ FVL/LNX では 2 値画像メモリと濃淡画像メモリのメモリ番号が統合されています。
例えば、2 値、濃淡メモリをそれぞれ 2 枚ずつ確保したとすると
(90X での動作)
2 値画像メモリ：0, 1
濃淡メモリ：0, 1
(FVL/LNX での動作)
2 値画像メモリ：0, 1
濃淡メモリ：2, 3
となります。90X では 2 値画像メモリと濃淡メモリの ID 番号は重複しておりましたが、FVL/LNX では画像メモリの ID 番号は重複しません。
特に 90X からの移植をする際にはカレントメモリ番号が 0 とは限りませんので、注意して移植してください。

903, 904

○ 2 値画像メモリを 32 面取ることが可能です。
ファイル名は 16 面以降「G000001?. SYS」(?=メモリ番号.16 進数)と自動的につけられます。

Lib_adrs_gray_memory

機　能

濃淡画像メモリのベース･ポインタ通知

形　式

```
#include "f_stdlib.h"
char *Lib_adrs_gray_memory ( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_alloc_gray_memory で確保した濃淡画像メモリ用ファイルのベースポインタを通知します。

① ***memory_no*** はメモリ番号.です。(0～15)
-1(CURRENT_ MEMORY)を指定した場合カレントメモリ用ファイルのアドレスを通知します。

戻り値

処理結果

値	意　味
0	異常終了しました。
0 以外	濃淡画像メモリのポインタを返します。

例

濃淡画像メモリをクリアします。

```
#include "f_stdlib.h"
#define LONG_WORD 4

void gray_memory_cls( )
{
    int *address;
    int i;
    int size;

    if ( NULL != (address = (void *)Lib_adrs_gray_memory ( CURRENT_ MEMORY )) )
    {
        size = ( Lib_get_fx_size () * Lib_get_fy_size () ) / LONG_WORD;

        for ( i = 0 ; i < size ; i++ )
            *address++ = 0x00000000;
    }
}
```

留意事項

○ カレントメモリとは画像入力･計測など画像メモリを処理対象とするライブラリの中でパラメータにメモリ番号.の指定がないライブラリの処理対象となる画像メモリです。
- ☆ Lib_freeze
- ☆ Lib_freerun
- ☆ Lib_memory_clear

Lib_adrs_bin_memory

機　能

2 値画像メモリのベース･ポインタ通知

形　式

```
#include "f_stdlib.h"
char *Lib_adrs_bin_memory ( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_alloc_bin_memory で取得した 2 値画像メモリ用ファイルのベースポインタを通知します。

① ***memory_no*** はメモリ番号.です。(0～15)
-1(CURRENT_ MEMORY)を指定した場合カレントメモリ用ファイルのアドレスを通知します。

戻り値

処理結果

値	意　　味
0	異常終了しました。
0 以外	2 値画像メモリのポインタです。

例

2 値画像メモリをクリアします。

```
#include "f_stdlib.h"
#define LONG_WORD      4
#define BIT_OF_MEMORY  8

void bin_memory_cls( )
{
    int *address;
    int i;
    int size;

    if ( NULL != (address = (void *)Lib_adrs_bin_memory
                                    ( CURRENT_ MEMORY )) )
    {
        size = ( Lib_get_fx_size() * Lib_get_fy_size() )
                                    / LONG_WORD / BIT_OF_MEMORY;

        for ( i = 0 ; i < size ; i++ )
             *address++ = 0x00000000;
    }
}
```

留意事項

○ カレントメモリとは画像入力・計測など画像メモリを処理対象とするライブラリの中でパラメータにメモリ番号.の指定がないライブラリの処理対象となる画像メモリです。

- ☆ Lib_freeze
- ☆ Lib_freerun
- ☆ Lib_memory_clear

Lib_free_gray_memory

機　能

濃淡画像メモリの解放

形　式

```
#include "f_stdlib.h"
int Lib_free_gray_memory( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_alloc_gray_memory で確保した濃淡画像メモリをメモリ用ファイル No.により解放します。
カレントメモリ用ファイルは解放できません。又、メモリ番号.0 も解放できません。

① ***memory_no*** はメモリ番号.です。(1～15)

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

メモリを新たに確保し、ビデオを入力します。
ビデオ入力後、何らかの処理を実行し、メモリを解放します。

```
#include "f_stdlib.h"
#include "f_video.h"

void free_gray_memory()
{
    int mem_no;
    int cmem_no;

    cmem_no=Lib_get_gray_memory();
    if ( ERROR_RETURN != (mem_no = Lib_alloc_gray_memory( )) )
    {
        if ( NORMAL_RETURN == Lib_change_gray_memory ( mem_no ) )
        {
            Lib_input_video_control( GRAY_PLANE );

            Lib_freeze( TRANSMIT );

            /*何らかの処理の実行*/

            Lib_change_gray_memory(cmem_no);
            Lib_free_gray_memory( mem_no );
        }
    }
}
```

留意事項

○ カレントメモリとは画像入力・計測など画像メモリを処理対象とするライブラリの中でパラメータにメモリ番号.の指定がないライブラリの処理対象となる画像メモリです。

- ☆ Lib_freeze
- ☆ Lib_freerun
- ☆ Lib_memory_clear

Lib_free_bin_memory

機　能

2 値画像メモリの解放

形　式

```
#include "f_stdlib.h"
int Lib_free_bin_memory( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_alloc_bin_memory で取得した 2 値画像メモリ用ファイルをメモリ番号.により解放します。カレントメモリ用ファイルは解放できません。又、メモリ番号.0 も解放できません。

① ***memory_no*** はメモリ番号.です。(1～15)

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

メモリを新たに確保し、ビデオを入力します。
ビデオ入力後、何らかの処理を実行し、メモリを解放します。

```
#include "f_stdlib.h"
#include "f_video.h"

void free_bin_memory()
{
    int mem_no;
    int cmem_no;

    cmem_no=Lib_get_bin_memory();
    if ( ERROR_RETURN != (mem_no = Lib_alloc_bin_memory( )) )
    {
        if ( NORMAL_RETURN == Lib_change_bin_memory ( mem_no ) )
        {
            Lib_input_video_control( BIN_PLANE );

            Lib_freeze( TRANSMIT );

            /*何らかの処理の実行*/

            Lib_change_bin_memory(cmem_no);
            Lib_free_bin_memory( mem_no );
        }
    }
}
```

留意事項

○ カレントメモリとは画像入力・計測など画像メモリを処理対象とするライブラリの中で
パラメータにメモリ番号.の指定がないライブラリの処理対象となる画像メモリです。

- ☆ Lib_freeze
- ☆ Lib_freerun
- ☆ Lib_memory_clear

Lib_get_gray_memory

機　能

濃淡画像メモリにおけるカレントメモリ番号.の参照

形　式

```
#include "f_stdlib.h"
int Lib_get_gray_memory( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリのカレントメモリ番号.を参照します。

戻り値

カレントメモリ番号.(0～15)を返します。

例

カレントメモリ番号.を参照し、その No.が 0 でない場合は 0 に変更します。

```
#include "f_stdlib.h"
#define MEM_NO 0

void set_memory( )
{
     if ( MEM_NO != Lib_get_gray_memory (  ))
         Lib_change_gray_memory ( MEM_NO );
}
```

留意事項

○ カレントメモリとは画像入力・計測など画像メモリを処理対象とするライブラリの中でパラメータにメモリ番号.の指定がないライブラリの処理対象となる画像メモリです。

- ☆ Lib_freeze
- ☆ Lib_freerun
- ☆ Lib_memory_clear

Lib_get_bin_memory

機　能

2 値画像メモリにおけるカレントメモリ番号.の参照

形　式

```
#include "f_stdlib.h"
int Lib_get_bin_memory( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

2 値画像メモリのカレントメモリ番号.を参照します。

戻り値

カレントメモリ番号.(0～15)を返します。

例

カレントメモリ番号.を参照し、その No.が 0 でない場合は 0 に変更します。

```
#include "f_stdlib.h"
#define MEM_NO 0

void set_memory( )
{
    if ( MEM_NO != Lib_get_bin_memory (  ))
        Lib_change_bin_memory ( MEM_NO );
}
```

留意事項

○ カレントメモリとは画像入力・計測など画像メモリを処理対象とするライブラリの中でパラメータにメモリ番号.の指定がないライブラリの処理対象となる画像メモリです。

- ☆ Lib_freeze
- ☆ Lib_freerun
- ☆ Lib_memory_clear

Lib_change_memory

機　能

グレイ&2値のカレントメモリ切り換え

形　式

```
#include "f_stdlib.h"
int Lib_change_memory ( int memory_no );
```

解　説

グレイ、2値のカレントメモリを変更します。

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

① ***memory_no*** はメモリ番号.です。(0～15)

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレントメモリを変更し、ビデオを入力します。

```
#include "f_stdlib.h"
#include "f_video.h"
#define MEM_NO 1

void new_memory()
{
    if ( NORMAL_RETURN == Lib_change_memory ( MEM_NO ) )
    {
        Lib_input_video_control( GRAY_PLANE|BIN_PLANE );

        Lib_freeze( TRANSMIT );
    }
}
```

留意事項

ありません。

Lib_change_gray_memory

機　能

濃淡画像メモリの切り換え

形　式

```
#include "f_stdlib.h"
int Lib_change_gray_memory ( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリのカレントメモリを変更します。

① ***memory_no*** はメモリ番号.です。(0～15)

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレントメモリを変更し、ビデオを入力します。

```
#include "f_stdlib.h"
#include "f_video.h"
#define MEM_NO 1

void new_memory()
{
    if ( NORMAL_RETURN == Lib_change_gray_memory ( MEM_NO ) )
    {
        Lib_input_video_control( GRAY_PLANE );

        Lib_freeze( TRANSMIT );
    }
}
```

留意事項

ありません。

Lib_change_bin_memory

機　能

2 値画像メモリの切り換え

形　式

```
#include "f_stdlib.h"
int Lib_change_bin_memory ( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

2 値画像メモリのカレントメモリを変更します。

① ***memory_no*** はメモリ番号.です。(0～15)

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレントメモリを変更し、ビデオを入力します。

```
#include "f_stdlib.h"
#include "f_video.h"
#define MEM_NO 1

void new_memory()
{
    if ( NORMAL_RETURN == Lib_change_bin_memory ( MEM_NO ) )
    {
        Lib_input_video_control( BIN_PLANE );

        Lib_freeze( TRANSMIT );
    }
}
```

留意事項

ありません。

Lib_get_fx_size

機　能

画像メモリの大きさ通知　横方向画素数

形　式

```
#include "f_stdlib.h"
int Lib_get_fx_size ( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

画像メモリの横方向画素数を参照します。
フレームバッファのサイズを取得するには Lib_get_dx_size を使用してください。

戻り値

横方向画素数(0～)を返します。

例

濃淡画像メモリをクリアします。

```
#include "f_stdlib.h"
#define LONG_WORD      4

void memory_cls( )
{
    int *address;
    int i;
    int size;

    if ( NULL != (address = (void *)Lib_adrs_bin_memory
                                        ( CURRENT_MEMORY )) )
    {
        size = ( Lib_get_fx_size() * Lib_get_fy_size() ) / LONG_WORD;

        for ( i = 0 ; i < size ; i++ )
             *address++ = 0x00000000;
    }
}
```

留意事項

○ 通常横方向は 512 ですが、m×n の拡張画像メモリ処理の初期化を実施すると通知される値は変更されます。
Lib_lsensor_init(ラインセンサカメラ処理の初期化)実行後、前記ライブラリで指定した水平取込み長が通知されます。
Lib_high_res_init、 Lib_high_res_xinit では、1230 と通知されます。

FVL/LNX

○ 90X では、現在のキャプチャサイズと同じ縦横サイズの画像メモリしか扱えませんでしたが、FVL/LNX ではそのような制限はありません。ユーザプログラム内で画像メモリサイズの取得に“Lib_get_fx_size”“Lib_get_fy_size”を使用している部分がある場合は、“Lib_get_gray_memory_size”または“Lib_get_bin_memory_size”で画像メモリサイズの取得をするように変更していただければ、現在のキャプチャサイズとは独立して画像メモリのサイズを取得できるようになります。
90X と同様に、現在のキャプチャサイズと同じ縦横サイズの画像メモリのみ扱う場合は、上記の変更は必要ありません。

Lib_get_fy_size

機　能

画像メモリの大きさ通知　縦方向画素数

形　式

```
#include "f_stdlib.h"
int Lib_get_fy_size ( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

画像メモリの縦方向画素数を参照します。
フレームバッファのサイズを取得するには Lib_get_dy_size を使用してください。

戻り値

縦方向画素数(0～)を返します。

例

濃淡画像メモリをクリアします。

```
#include "f_stdlib.h"
#define LONG_WORD      4

void memory_cls( )
{
    int *address;
    int i;
    int size;

    if ( NULL != (address = (void *)Lib_adrs_bin_memory (CURRENT_MEMORY)) )
    {
        size = ( Lib_get_fx_size() * Lib_get_fy_size() ) / LONG_WORD;

        for ( i = 0 ; i < size ; i++ )
            *address++ = 0x00000000;
    }
}
```

留意事項

○ 通常縦方向は 480 ですが、m×n の拡張画像メモリ処理の初期化を実施すると通知される値は変更されます。
Lib_lsensor_init(ラインセンサカメラ処理の初期化)実行後、前記ライブラリで指定した垂直取込み長が通知されます。
Lib_high_res_init、 Lib_high_res_xinit では、1030 と通知されます。

FVL/LNX

○ 90X では、現在のキャプチャサイズと同じ縦横サイズの画像メモリしか扱えませんでしたが、FVL/LNX ではそのような制限はありません。ユーザプログラム内で画像メモリサイズの取得に“Lib_get_fx_size”“Lib_get_fy_size”を使用している部分がある場合は、“Lib_get_gray_memory_size”または“Lib_get_bin_memory_size”で画像メモリサイズの取得をするように変更していただければ、現在のキャプチャサイズとは独立して画像メモリのサイズを取得できるようになります。
90X と同様に、現在のキャプチャサイズと同じ縦横サイズの画像メモリのみ扱う場合は、上記の変更は必要ありません。

Lib_get_dx_size

機　能

フレームバッファの大きさ通知　横方向画素数

形　式

```
#include "f_stdlib.h"
int Lib_get_dx_size ( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フレームバッファの横方向画素数を参照します。

戻り値

横方向画素数(0～)

例

画面サイズの線を引きます。

```
#include "f_stdlib.h"
#include "f_graph.h"

void draw_line( )
{
    int xs, xe;
    int ys, ye;

    xs = 0;
    xe = Lib_get_dx_size() - 1;
    ys = 0;
    ye = Lib_get_dy_size() - 1;

    Lib_line( xs, ys, xe, ye, SOLID_LINE );
    Lib_line( xe, ys, xs, ye, SOLID_LINE );
}
```

留意事項

○ m×n の拡張画像メモリ(ラインセンサなど)を使用しない場合は
Lib_get_fx_size と Lib_get_dx_size は同じ値を返します。

Lib_get_dy_size

機　能

フレームバッファの大きさ通知　縦方向画素数

形　式

```
#include "f_stdlib.h"
int Lib_get_dy_size ( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フレームバッファの縦方向画素数を参照します。

戻り値

縦方向画素数(0～)

例

画面サイズの線を引きます。

```
#include "f_stdlib.h"
#include "f_graph.h"

void draw_line( )
{
    int xs, xe;
    int ys, ye;

    xs = 0;
    xe = Lib_get_dx_size() - 1;
    ys = 0;
    ye = Lib_get_dy_size() - 1;

    Lib_line( xs, ys, xe, ye, SOLID_LINE );
    Lib_line( xe, ys, xs, ye, SOLID_LINE );
}
```

留意事項

○ m×nの拡張画像メモリ(ラインセンサなど)を使用しない場合は
Lib_get_fy_size と Lib_get_dy_size は同じ値を返します。

Lib_malloc

機　能

メモリブロックの割当

形　式

```
#include "f_stdlib.h"
char *Lib_malloc ( size_t memory_size );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

確保要求分のメモリブロックを割り当て、その領域の先頭にあたるポインタを返します。
割り当てるメモリブロックはヒープ領域に確保します。
901 の場合は、1 ブロック当たり 64 バイトで 1024 ブロック分を用意しています。
従って、確保要求サイズが大きい場合には後述の Lib_mlalloc を使用してください。
902 の場合は、使用可能なメモリが許す限り使用できます。

① ***memory_size*** は確保したいメモリサイズ(バイト)を指定する。

戻り値

処理結果

値	意　味
0	異常終了しました。
0 以外	メモリプロックへのポインタです。

例

メモリを確保し、何らかの処理を実行後、メモリを解放します。

```
#include "f_stdlib.h"
#define  ALLOC_SIZE  128

void work_memory()
{
    char *ptr;

    if ( NULL != ( ptr = Lib_malloc( ALLOC_SIZE )) )
    {
        /*何らかの処理の実行*/
        Lib_free( ptr );
    }
}
```

留意事項

○ 901,903,904 の場合、Lib_mlalloc との相違は次の通りです。

メモリブロックの割当ライブラリ	1 ブロック当りのバイト数	ヒープ領域バイト数	対応する解放関数
Lib_malloc	64	64K	Lib_free
Lib_mlalloc	4096	メモリの許す限り	Lib_lfree

○ 902 の場合は、Lib_malloc と Lib_mlalloc は、全く同じになります。

Lib_mlalloc

機　能

メモリブロックの割当Ⅱ

形　式

```
#include "f_stdlib.h"
char *Lib_mlalloc ( size_t memory_size );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

確保要求分のメモリブロックを割り当て、その領域の先頭にあるポインタを返します。
割り当てるメモリブロックはヒープ領域に確保します。

① ***memory_size*** は確保したいメモリサイズ(バイト)を指定する。

戻り値

処理結果

値	意　味
0	異常終了しました。
0 以外	メモリブロックへのポインタです。

例

メモリを確保し、何らかの処理を実行後、メモリを解放します。

```
#include "f_stdlib.h"
#define  ALLOC_SIZE  0x80000

void work_memory()
{
    char *ptr;

    if ( NULL != ( ptr = Lib_mlalloc( ALLOC_SIZE )) )
    {
        /*何らかの処理の実行*/

        Lib_lfree( ptr );
    }
}
```

留意事項

○ 901,903,904 の場合、1 ブロック当り 4096 バイトでメモリの許す限り割り当てます。
○ 901,903,904 の場合、Lib_malloc との相違は次の通りです。

メモリブロックの割当ライブラリ	1 ブロック当りのバイト数	ヒープ領域バイト数	対応する解放関数
Lib_malloc	64	64K	Lib_free
Lib_mlalloc	4096	メモリの許す限り	Lib_lfree

○ 902 の場合は、Lib_malloc と Lib_mlalloc は、全く同じになります。

Lib_free

機　能

メモリブロックの解放

形　式

```
#include "f_stdlib.h"
void Lib_free( char *ptr )
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

メモリブロックをヒープ領域から解放します。

① ***ptr*** は解放するメモリブロックへのポインタを指定します。
901,903 の場合、ポインタは必ず Lib_malloc で返されたポインタでなければなりません。
902 の場合は、Lib_malloc, Lib_mlalloc で返されたポインタが両方とも使用できます。

戻り値

ありません。

例

メモリを確保し、何らかの処理を実行後、メモリを解放します。

```
#include "f_stdlib.h"
#define  ALLOC_SIZE  128

void work_memory()
{
    char *ptr;

    if ( NULL != ( ptr = Lib_malloc( ALLOC_SIZE )) )
    {
        /*何らかの処理の実行*/

        Lib_free( ptr );
    }
}
```

留意事項

○ 901,903,904 の場合、Lib_mlalloc で返されたポインタを入力しても何ら処理を行いません。

Lib_lfree

機　能

メモリブロックの解放Ⅱ

形　式

```
#include "f_stdlib.h"
void Lib_lfree( char *ptr )
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

メモリブロックをヒープ領域から解放します。

① ***ptr*** は解放するメモリブロックへのポインタを指定します。
901,903 の場合、ポインタは必ず Lib_mlalloc で返されたポインタでなければなりません。
902 の場合は、Lib_malloc, Lib_mlalloc で返されたポインタが両方とも使用できます。

戻り値

ありません。

例

メモリを確保し、何らかの処理を実行後、メモリを解放します。

```
#include "f_stdlib.h"
#define  ALLOC_SIZE  0x80000

void work_memory()
{
    char *ptr;

    if ( NULL != ( ptr = Lib_mlalloc( ALLOC_SIZE )) )
    {
        /*何らかの処理の実行*/

        Lib_lfree( ptr );
    }
}
```

留意事項

○ 901,903,904 の場合、Lib_malloc で返されたポインタを入力しても何ら処理を行いません。

Lib_get_memory_type

機　能

メモリタイプの取得

形　式

```
#include "f_stdlib.h"
int Lib_get_memory_type( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

現在の画像メモリタイプを参照します。

戻り値

現在のメモリタイプが返ります。

値	定　　数	意　　味
0	NORMAL_MEMORY_TYPE	通常の 512×480 モードです。
1	EXTEND_MEMORY_TYPE	m×n の拡張画像メモリモードです。

例

Lib_get_adrs_work_memory を参照してください。

留意事項

ありません。

Lib_get_adrs_work_memory

機　能

拡張画像メモリ時用のワークメモリアドレスの取得

形　式

```
#include "f_video.h"
unsigned char *Lib_get_adrs_work_memory( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

m×n 時のワーク･メモリ･アドレスを参照します。
このメモリはメモリ内容をフレームバッファに転送する際のワーク･メモリとして使用します。
サイズは 512×480 バイト確保しています。

戻り値

ワーク･メモリ･アドレスのポインタです。

例

濃淡画像拡張メモリをワーク･メモリに転送し、モニタ TV に表示します。

```
#include "f_stdlib.h"
#include "f_video.h"

void gray_memory_move( int memory_no )
{
   unsigned char *lmem_top_adrs, *mem_top_adrs, *lwork_adrs, *work_adrs;
    int fx_size, fy_size;      /*メモリのサイズ*/
    int dx_size, dy_size;      /*フレームバッファのサイズ*/
    int x, y;

    /*フレームバッファ･サイズの取得*/
    dx_size = Lib_get_dx_size();      dy_size = Lib_get_dy_size();
    /*メモリ･サイズの取得*/
    fx_size = Lib_get_fx_size();      fy_size = Lib_get_fy_size();

    /*m×n 対応のメモリか確認/
    if( NORMAL_MEMORY_TYPE != Lib_get_memory_type() )
    {
        /*画像メモリのアドレス取得*/
        if( (unsigned char *)NULL != (lmem_top_adrs =
                (unsigned char *)Lib_adrs_gray_memory ( memory_no )) )
        {
            /*フレームバッファ転送用メモリのアドレス取得*/
            mem_top_adrs  = Lib_get_adrs_work_memory( );

            for( y = 0; y < dy_size; y++ )
            {
                lwork_adrs = lmem_top_adrs;
                work_adrs  = mem_top_adrs;
                for( x = xs; x < dx_size ; x++ )
                    *work_adrs++ = *lwork_adrs++;
                lmem_top_adrs += fx_size;
                mem_top_adrs  += dx_size;
            }
            Lib_ex_disp_gray_memory();
        }
    }
}
```

留意事項

○ m×n の拡張画像メモリ(ラインセンサなど)モード以外の時にコールした場合は 0 が返ります。

Lib_gray_transmit

機　能

濃淡フレームバッファへの矩形領域画像転送

形　式

```
#include "f_stdlib.h"
int Lib_gray_transmit( int memory_no, int xs, int ys, int xe, int ye );
```

901	902	903	904	FVL/LNX	高分解能
△	○	○	○	○	○

解　説

① ***memory_no*** はメモリ番号.です。

② ***xs*** は開始 X 座標です。

③ ***ys*** は開始 Y 座標です。

④ ***xe*** は終了 X 座標です。

⑤ ***ye*** は終了 Y 座標です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

```
#include "f_stdlib.h"

#define MEMORY_NO         0
#define XS              100
#define YS              100
#define XE              300
#define YE              300

void box_transmit()
{
    Lib_input_video_control( GRAY_PLANE );

    Lib_freeze( NOT_TRANSMIT );

    Lib_gray_transmit( MEMORY_NO, XS, YS, XE, YE );
}
```

留意事項

○ なお、本ライブラリは 901,901a,901SB ではサポートしておりません。

Lib_buff_to_memory

機　能

連続画像バッファの画像メモリへの転送

形　式

```
#include "f_stdlib.h"
int Lib_buff_to_memory( char *address, int no, int flag, int mem_no );
```

901	902	903	904	FVL/LNX	高分解能
△	△	○	○	○	○

解　説

本ライブラリは Lib_xframe_chain_freeze や Lib_xcamera_chain_freeze で取得した連続画像バッファ内の任意の 1 フレームを指定画像メモリに転送します。

① ***address*** は連続画像バッファの先頭アドレスです。

② ***no*** は転送フレーム番号(0～)

③ ***flag*** は転送メモリ種別です。(下記の一方を選択。)

値	定　数	意　味
0x01	BIN_PLANE	2 値画像メモリへ転送します。
0x20	GRAY_PLANE	濃淡画像メモリへ転送します。

④ ***memory_no*** はメモリ番号. です。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

Lib_xframe_chain_freeze の例を参照してください。

留意事項

○ 901 の場合
使用できる機種は 901b/901c/901SB2/901NT に限定されます。

○ 902 の場合
キャプチャボードが RICE および FVC01 に限定されます。

Lib_get_gray_memory_size

機　能

濃淡画像メモリの大きさ取得

形　式

```
#include "f_stdlib.h"
int Lib_get_gray_memory_size( int mem_no, int *fxsize,
                                                    int *fysize, int *memsize );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

指定された濃淡画像メモリの横方向画素数、縦方向画素数、画像メモリサイズ(byte単位)を取得します。

① ***mem_no*** はメモリ番号です。

② ****fxsize*** は画像メモリの横方向画素数へのポインタです。

③ ****fysize*** は画像メモリの縦方向画素数へのポインタです。

④ ****memsize*** は画像メモリサイズ(byte単位) へのポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

濃淡画像メモリ確保後、サイズを取得します。

```
#include "f_stdlib.h"

void func( void )
{
    int         mem_no;
    int         fxsize, fysize, memsize;

    if( 0 <= ( mem_no = Lib_alloc_gray_memory() ) )
    {
        Lib_get_gray_memory_size( mem_no, &fxsize, &fysize, &memsize );

        /***** 何らかの処理 *****/

    }
}
```

留意事項

出力先のポインタにNULLを設定した場合は、出力されません。

Lib_get_bin_memory_size

機　能

2値画像メモリの大きさ取得

形　式

```
#include "f_stdlib.h"
int Lib_get_bin_memory_size( int mem_no, int *fxsize, int *fysize, int *memsize );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

指定された2値画像メモリの横方向画素数、縦方向画素数、画像メモリサイズ(byte単位)を取得します。

① ***mem_no*** はメモリ番号です。

② ****fxsize*** は画像メモリの横方向画素数へのポインタです。

③ ****fysize*** は画像メモリの縦方向画素数へのポインタです。

④ ****memsize*** は画像メモリサイズ(byte単位) へのポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-10004	ERR_INVALID_ID	指定されたビデオウィンドウは存在しません。

例

2値画像メモリ確保後、サイズを取得します。

```
#include "f_stdlib.h"

void func( void )
{
    int  mem_no;
    int  fxsize, fysize, memsize;

    if( 0 <= ( mem_no = Lib_alloc_bin_memory() ) )
    {
        Lib_get_bin_memory_size( mem_no, &fxsize, &fysize, &memsize );

        /***** 何らかの処理 *****/

    }
}
```

留意事項

○ 出力先のポインタにNULLを設定した場合は、出力されません。

4. メモリ操作ライブラリ

本ライブラリはフレームバッファ、画像メモリをクリアするライブラリとなっています。
2値、濃淡画像のメモリ操作(膨張、微分など)は、各々のライブラリに納められています。

Lib_memory_clear

機　能

メモリのクリア

形　式

```
#include "f_image.h"
void Lib_memory_clear( int plane );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定されたフレームバッファのカレントメモリとフレームバッファをクリアします。

① ***plane*** はフレームバッファです。
（該当するメモリのビットをオンにしたビットパタンで指定します）

値	定　数	意　味
0x01	BIN_PLANE	2 値用フレームバッファ、メモリともにクリアします。
0x20	GRAY_PLANE	濃淡用フレームバッファ、メモリともにクリアします。
0x40	LINE_PLANE	線画用フレームバッファをクリアします。
0x80	CHAR_PLANE	文字用フレームバッファをクリアします。

戻り値

ありません。

例

フレームバッファをクリア後、モニタに文字を表示します。

```
#include "f_image.h"

void clr_char()
{
    Lib_memory_clear( LINE_PLANE|CHAR_PLANE );

    Lib_chrdisp( 10, 10, "SAMPLE" );
}
```

留意事項

ありません。

Lib_bin_memory_cls

機　能

2 値画像メモリクリア

形　式

```
#include "f_image.h"
int Lib_bin_memory_cls( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

2 値画像メモリをクリアします。

① ***memory_no*** はメモリ番号.です。
メモリ番号.に -1(CURRENT_MEMORY)を指定した場合はカレントメモリをクリアします。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

メモリをクリア後、ビデオを入力します。

```
#include "f_image.h"
#include "f_video.h"

void cls_in()
{
    Lib_bin_memory_cls( CURRENT_MEMORY );

    Lib_freeze( TRANSMIT );
}
```

留意事項

○ この関数を実行してもフレームバッファ上は何ら変化はありません。
つまり、モニタ TV の濃淡画像表示は変化しません。

Lib_gray_memory_cls

機　能

濃淡画像メモリクリア

形　式

```
#include "f_image.h"
int Lib_gray_memory_cls( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリをクリアします。

① ***memory_no*** はメモリ番号.です。
メモリ番号.に-1(CURRENT__MEMORY)を指定した場合はカレントメモリをクリアします。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

メモリをクリア後、ビデオを入力します。

```
#include "f_image.h"
#include "f_video.h"

void cls_in()
{
    Lib_gray_memory_cls( CURRENT_MEMORY );

    Lib_freeze( TRANSMIT );
}
```

留意事項

○ この関数を実行してもフレームバッファ上は何ら変化はありません。
つまり、モニタ TV の濃淡画像表示は変化しません。

5. 画像計測ライブラリ

本ライブラリは画像メモリを計測するものです。

Lib_histogram

機　能

ヒストグラムの計算

形　式

```
#include "f_bgray.h"
int Lib_histogram( int memory_no, struct HISTDATA *histp );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

処理領域内の濃度値(0～255)のヒストグラム分布を計算するものです。
本ライブラリでは、濃度毎の頻度値の他に

☆頻度最大値 ・・・・・・・・・・・・・・・・・頻度値の最大値
☆頻度最小値 ・・・・・・・・・・・・・・・・・頻度値の最小値
☆最大頻度濃度値 ・・・・・・・・・・・・・・頻度最大値に対応する濃度値
☆最小頻度濃度値 ・・・・・・・・・・・・・・頻度最小値に対応する濃度値

を同時に計算することができます。

① ***memory_no*** はメモリ番号.です。

② ****histp*** は構造体 HISTDATA のポインタです。

もとめた値は histp が指す構造体 HISTDATA 内に格納します。
構造体 HISTDATA は f_bgray.h 内で次のように宣言されています。

```
#define HIST_LEVEL          256

struct HISTDATA
{
    unsigned int max_frequency;             /* 頻度値の最大値              */
    unsigned int min_frequency;             /* 頻度値の最小値              */
    unsigned int max_fre_level;             /* 頻度最大値に対応する濃度値  */
    unsigned int min_fre_level;             /* 頻度最小値に対応する濃度値  */
    unsigned int hist_tbl[HIST_LEVEL];      /* 各濃度値に対する頻度        */
};
```

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

ヒストグラムを計測し、結果を表示します。

```
#include  "f_stdio.h"
#include  "f_graph.h"
#include  "f_bgray.h"


void disp_hist()
{
    static char string[40];
    static struct HISTDATA histp;

    if( NORMAL_RETURN == Lib_histogram( Lib_get_gray_memory(), &histp ) )
    {
        Lib_sprintf( string, "最大頻度濃度値:%3u  頻度:%6u",
                              histp.max_fre_level,histp.max_frequency );
        Lib_chrdisp( 1, 1, string );
        Lib_sprintf( string, "最小頻度濃度値:%3u  頻度:%6u",
                              histp.min_fre_level, histp.min_frequency );
        Lib_chrdisp( 1, 2, string );
    }
}
```

留意事項

ありません。

Lib_histogram_step

機　能

ヒストグラムの計算(1 画素飛び)

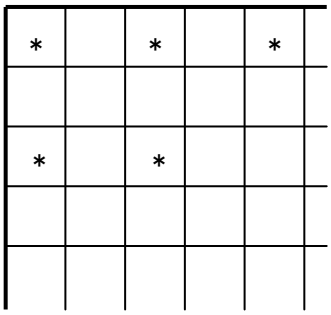
形　式

```
#include "f_bgray.h"
int Lib_histogram_step( int memory_no, struct HISTDATA *histp );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

処理領域内の濃度値(0～255)のヒストグラム分布を計算するものです。
前項の Lib_histogram との相違は処理時間を短縮するために XY 座標ともに偶数の画素のみを計測している事です。
1 画素飛びに計測しているために処理時間は 4 分の 1 になっています。



※ ＊は計測画素を表わします。

本ライブラリでは、濃度毎の頻度値の他に
- ☆頻度最大値 ····· 頻度値の最大値
- ☆頻度最小値 ····· 頻度値の最小値
- ☆最大頻度濃度値 ····· 頻度最大値に対応する濃度値
- ☆最小頻度濃度値 ····· 頻度最小値に対応する濃度値を同時に計算することができます。

① ***memory_no*** はメモリ番号. です。

② ***histp*** は構造体 HISTDATA のポインタです。

もとめた値は histp が指す構造体 HISTDATA 内に格納します。
構造体 HISTDATA は f_bgray.h 内で次のように宣言されています。

```
#define HIST_LEVEL          256

struct HISTDATA
{
    unsigned int max_frequency;          /* 頻度値の最大値                    */
    unsigned int min_frequency;          /* 頻度値の最小値                    */
    unsigned int max_fre_level;          /* 頻度最大値に対応する濃度値        */
    unsigned int min_fre_level;          /* 頻度最小値に対応する濃度値を
                                            同時に計算することができます。    */
    unsigned int hist_tbl[HIST_LEVEL];   /* 各濃度値に対する頻度              */
};
```

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

ヒストグラムを計測し、結果を表示します。

```
#include  "f_stdio.h"
#include  "f_graph.h"
#include  "f_bgray.h"

void disp_hist()
{
    static char string[40];
    static struct HISTDATA histp;

    if( NORMAL_RETURN == Lib_histogram_step( Lib_get_gray_memory(), &histp ) )
    {
        Lib_sprintf( string, "最大頻度濃度値:%3u  頻度:%6u",
                             histp.max_fre_level, histp.max_frequency );
        Lib_chrdisp( 1, 1, string );
        Lib_sprintf( string, "最小頻度濃度値:%3u  頻度:%6u",
                             histp.min_fre_level, histp.min_frequency );
        Lib_chrdisp( 1, 2, string );
    }
}
```

留意事項

ありません。

Lib_display_histogram

機　能

ヒストグラムデータの表示

形　式

```
#include "f_bgray.h"
int Lib_display_histogram( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

モニタ TV 上に表示されているヒストグラムデータを見ながら、トラックボール(又はキー)を使用して濃度値を決めるライブラリです。
ヒストグラムの計測は前項の Lib_histogram を使用しています。

① ***memory_no*** はメモリ番号.です。

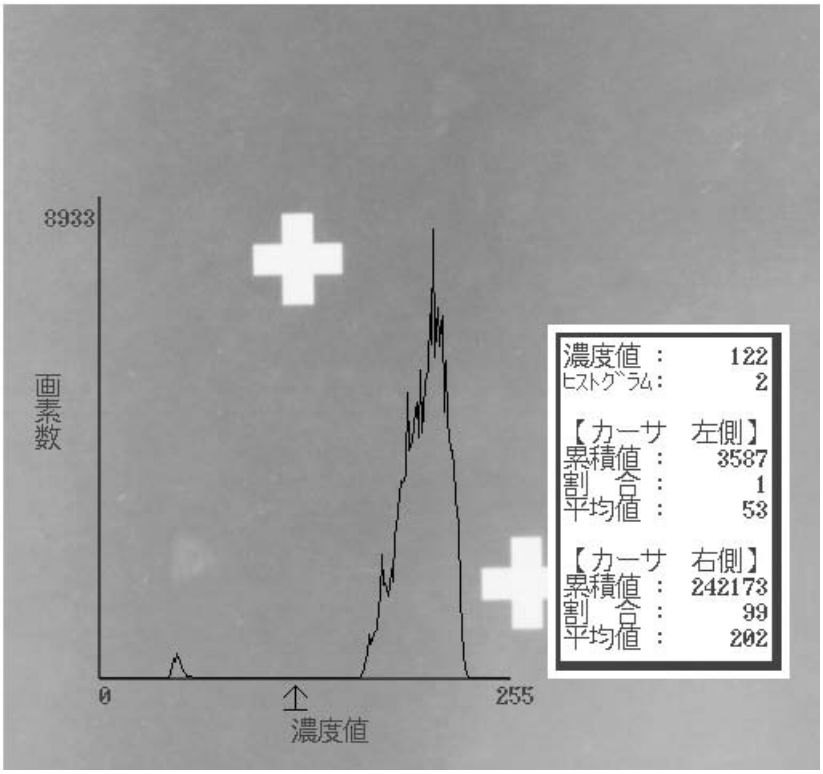
戻り値

濃度値

値	定　　数	意　　味
0～	ありません	濃度値です。
-1	ERROR_RETURN	異常終了しました。

操作方法

①モニタ TV 表示



※ この画像は反転表示しています。

②ヒストグラム表示内容

濃度値(Level): aa ←現在のカーサ位置の濃度値
ヒストグラム(Hist):bbbbbb ←現在のカーサ位置のヒストグラム値

【カーサ 左側】(CURSOR LEFT)
累積値(Sum):cccccc ←自分を含めて左側にあるヒストグラムの累積値、全体に占める割合(%)
割 合(%): dd ←平均濃度値
平均値(Average): eee

【カーサ 右側】(CURSOR RIGHT)
累積値(Sum):ffffff ←自分より右側にあるヒストグラムについての同上値
割 合(%): gg
平均値(Average): hhh

※ ()は英語モード時の表示内容

③トラックボール操作
　トラックボールを左に移動(左矢印キー押下) ･･･････ カーサ位置が左に移動(→0)
　トラックボールを右に移動(右矢印キー押下) ･･･････ カーサ位置が右に移動(→255)
　左ボタンを押す (EXECUTE キー押下) ･･･････････････ 操作終了

例

ヒストグラムデータを表示し、2 値化しきい値を決め、表示します。

```
#include  "f_gui.h"
#include  "f_video.h"
#include  "f_stdlib.h"
#include  "f_bgray.h"


void set_level()
{
    int level;

    Lib_input_video_control( GRAY_PLANE );
    Lib_display_control( (GRAY_PLANE|LINE_PLANE|CHAR_PLANE));

    Lib_init_cursor();
    Lib_freeze( TRANSMIT );
    if( ERROR_RETURN != (level = Lib_display_histogram
                                    ( Lib_get_gray_memory() )) )
    {
        Lib_input_video_control( BIN_PLANE );
        Lib_display_control( (BIN_PLANE|LINE_PLANE|CHAR_PLANE));
        Lib_video_bin_level( level );
        Lib_freeze( TRANSMIT );
        Lib_display_keyinput( 10,450, "入力待ち");
        Lib_input_video_control( GRAY_PLANE );
        Lib_display_control( (GRAY_PLANE|LINE_PLANE|CHAR_PLANE));
    }
}
```

留意事項

○ Lib_init_cursor を実行前にコールしてください。
○ Lib_freeze をライブラリ内ではコールしていませんので、メモリ内容と表示内容が異なる場合、結果が異常になります。ご留意ください。

Lib_ascope

機　能

Ａスコープ表示

形　式

```
#include "f_bgray.h"
void Lib_ascope( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

Ｘ断面、Ｙ断面での濃度値をモニタ TV 上に表示するもので、Ｘおよび Ｙ座標をトラックボール(4 方向キー)を使用して調整します。

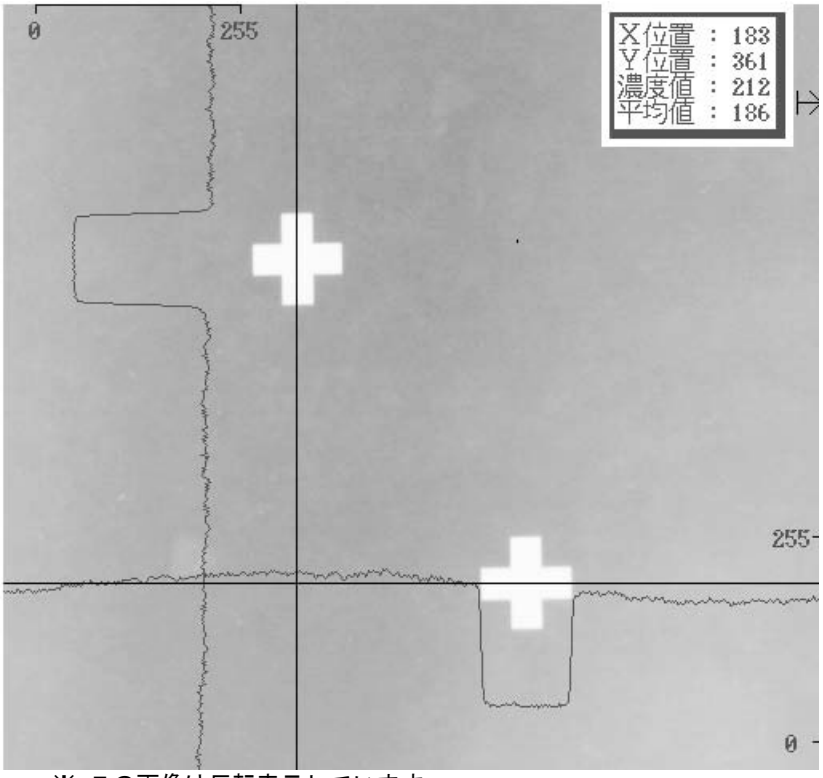
① ***memory_no*** はメモリ番号. です。

戻り値

ありません。

操作方法

①モニタ TV 表示



※ この画像は反転表示しています。

② A スコープ表示内容

```
X 位置( X pos ):  aaa      ←現在の X 座標位置
Y 位置( Y pos ):  bbb      ←現在の Y 座標位置
濃度値( Level ):  ccc      ←現在の座標位置の濃度値
平均値(Average):  ddd      ←XY 線下の平均濃度
      ※ () は英語モード時の表示内容
```

③トラックボール操作

トラックボールを 4 方向に移動(4 方向キー押下) ····· XY 座標位置が 4 方向に移動
左ボタンを押す(EXECUTE キー押下) ····· 操作終了

例

A スコープを表示します。

```
#include  "f_gui.h"
#include  "f_video.h"
#include  "f_bgray.h"

void set_level()
{
    Lib_input_video_control( GRAY_PLANE );
    Lib_display_control( (GRAY_PLANE|LINE_PLANE|CHAR_PLANE));

    Lib_init_cursor();
    Lib_freeze( TRANSMIT );
    Lib_ascope( Lib_get_gray_memory() );
    Lib_freerun();
}
```

留意事項

○ Lib_init_cursor を実行前にコールしてください。
○ Lib_freeze をライブラリ内ではコールしていませんので、メモリ内容と表示内容が異なる場合、結果が異常になります。ご留意ください。

機　能

濃度値鳥瞰図描画

形　式

```
#include "f_bgray.h"
int Lib_g3dwire( int memory_no, int mode, int inter,
                                 int drw[ ], int color, int p_opt );
```

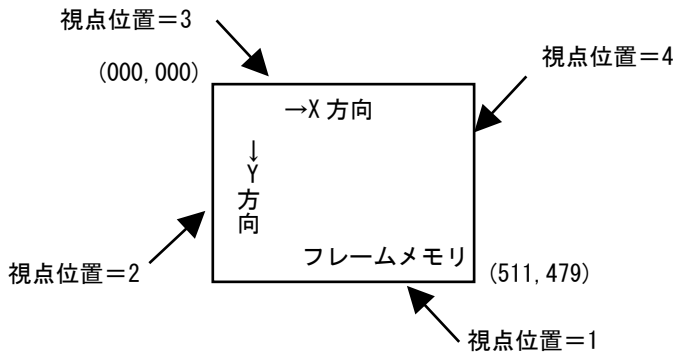
901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

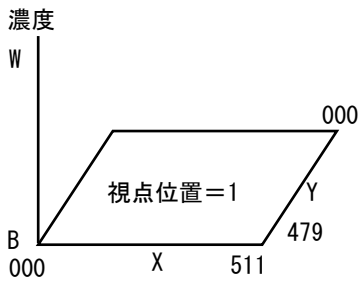
処理領域内の 8 ビット濃淡画像において、フレームメモリを水平面にして、各点の濃度値を高さとした、Wire_frame 形状の鳥瞰図を、線画メモリに表示します。
なお、トラックボールを操作して、鳥瞰図の指点位置および、描画サイズを変更する事ができます。

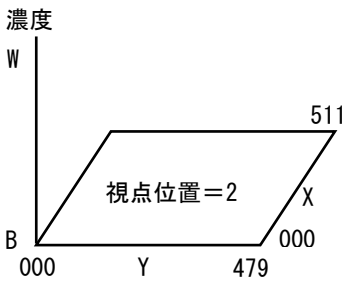
① ***memory_no*** はメモリ番号. です。

② ***mode*** は指点位置(View)です。（初期値：3）
最初に鳥瞰図を描く位置を 1 から 4 の数値で指定します。



各視点位置に対応したモニタ TV 表示形式を下記に示します。





③ ***inter*** は描画間隔です。
Wire 形状を描く間隔を、水平面の X-Y 座標単位で 1 から 50 の数値で指定します。
（フレームメモリのデータをサンプリングする間隔です。）

④ ***drw[ ]*** は描画サイズ情報格納アドレスです。
下記の形式で描画サイズ情報が格納されているアドレスを指定します。

a. 高さ方向縮小率
高さ方向 1 画素に対応する濃度値を 1 または、2 の数値で指定します。
1 を指定すると濃度値＝100 が 100 画素の高さで
2 を指定すると濃度値＝100 が 50 画素の高さで、
表示されます。

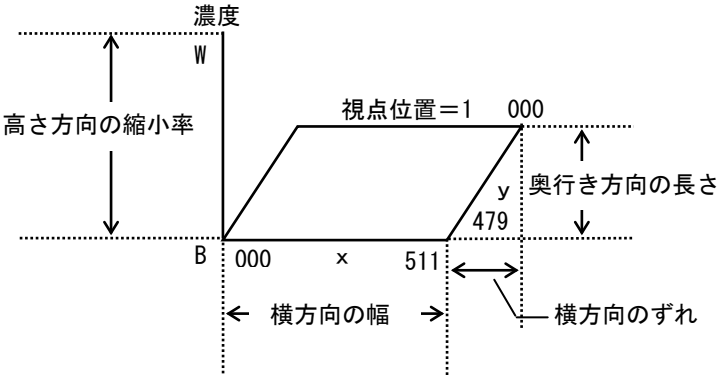
ワード位置

				視点位置 1 または 3	視点位置 2 または 4
P→	+0	高さ方向の縮小率	デフォルト値	1	1
	+1	横方向の幅	——————→	256	240
	+2	奥行き方向の長さ		120	128
	+3	横方向のずれ		192	180

b. 横方向の幅
水平面の横方向の幅を 128～394 の画素数単位で指定します。

c. 奥行き方向の長さ
水平面の奥行き方向の長さを画素数単位で指定します。
指定可能な数値範囲は
高さ方向縮小率が 1 の場合は 80～156、
高さ方向縮小率が 2 の場合は 80～284 です。

d. 横方向のずれ
水平面の手前の辺と、奥の辺のずれを 48～(442－横方向の幅)の画素数単位で指定します。



⑤ ***color*** は濃度値です。

値	定　　数	意　　味
0	ありません。	濃度値 255(W)を上に表示します。
1	ありません。	濃度値 0(B)を上に表示します。

⑥ ***p_opt*** はパッド操作オプションです。

値	定　　数	意　　味
0	ありません。	操作パッドを表示しません。
1	ありません。	操作パッドを表示します。

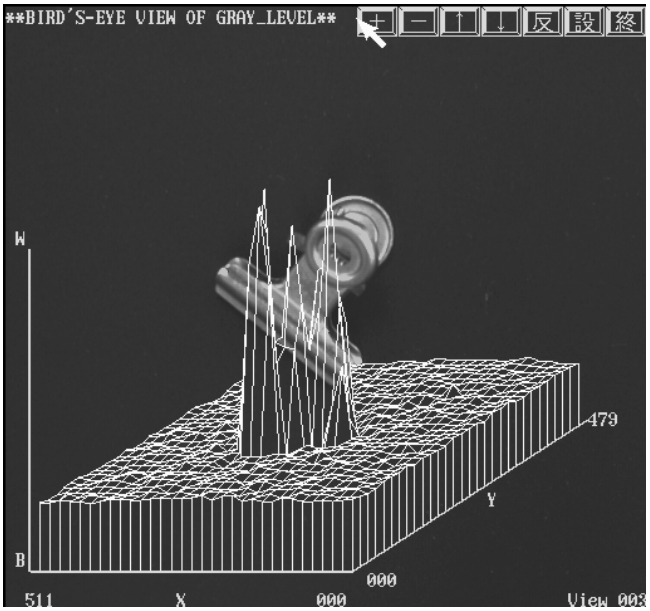
戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

操作方法

① モニタ TV 表示



② 濃度値鳥瞰図描画表示内容
　View aaa ･･･ 視点位置

③ トラックボール操作(パッド操作オプションを 1 で指定した時のみ)
○ +‥‥‥‥‥ 視点位置を 1 加算します。(4 の場合は 1 になります。)
○ −‥‥‥‥‥ 視点位置を 1 減算します。(1 の場合は 4 になります。)
○↑‥‥‥‥‥ 現在濃度値が 2 の場合、濃度値を 1 で表示します。
○↓‥‥‥‥‥ 現在濃度値が 1 の場合、濃度値を 2 で表示します。
○反(Tr)‥‥‥ 現在表示されている鳥瞰図を反転
(0→255 または, 255→0)して表示します。
○設(St)‥‥‥ パラメータの設定を行います。
○指点位置(View)
最初に鳥瞰図を描く位置を 1 から 4 の数値で指定します。

○描画間隔(Space)
Wire 形状を描く間隔を、水平面の X-Y 座標単位で 1 から 50 の数値で指定します。(フレームメモリのデータをサンプリングする間隔です。)

○高さ方向縮小率 (Reduce)
高さ方向 1 画素に対応する濃度値を 1 または、2 の数値で指定します。
1 を指定すると濃度値＝100 が 100 画素の高さで
2 を指定すると濃度値＝100 が 50 画素の高さで、
表示されます。

○横方向の幅(Wide)
水平面の横方向の幅を 128～394 の画素数単位で指定します。

○奥行き方向の長さ (Length)
水平面の奥行き方向の長さを画素数単位で指定します。
指定可能な数値範囲は
高さ方向縮小率が 1 の場合は 80～156
高さ方向縮小率が 2 の場合は 80～284 です。

○横方向のずれ(Gap)
水平面の手前の辺と、奥の辺のずれを 48～(442－横方向の幅)の画素数単位で指定します。
[注] 上記の各パラメータ値が範囲外の場合は、「442-横方向の幅」で計算された値が設定されます。

○終(Qu) ‥‥‥ 操作終了。

④ システムパラメータ「処理領域」
現在使用中のビデオ入力チャネル側の「処理領域」で示される、矩形領域の内側のみが処理領域となります。

※ ()は英語モード時の表示内容

例

メモリを確保後、画像を入力し、鳥瞰図を表示します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_gray.h"

void g3dwire( void )
{
    int mem_no;
    int drw[4];
    int mode;
    int inter;
    int color;

    if( ERROR_RETURN != ( mem_no = Lib_alloc_gyar_memory() ) )
    {
        if( NORMAL_RETURN == Lib_change_gray_memory( mem_no ) )
        {
            Lib_input_video_control( GRAY_PLANE );
            Lib_freeze( TRANSMIT );
            mode   = 3;
            inter  = 16;
            color  = 0;
            drw[0] = 1;
            drw[1] = 256;
            drw[2] = 120;
            drw[3] = 192;
            Lib_g3dwire( mem_no, mode, inter, drw, color, 0 );
            Lib_change_gray_memory( 0 )
        }
        Lib_free_gray_memory( mem_no );
    }
}
```

留意事項

○ Lib_freeze を実行前にコールしてください。

○ パッド操作オプションを 0 で指定した場合、鳥瞰図を表示後、処理を終了しますので何らかの処理を行ってください。

Lib_ex_ascope

機　能

拡張画像メモリ時用Ａスコープ表示

形　式

```
#include "f_bgray.h"
void Lib_ex_ascope( int memory_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

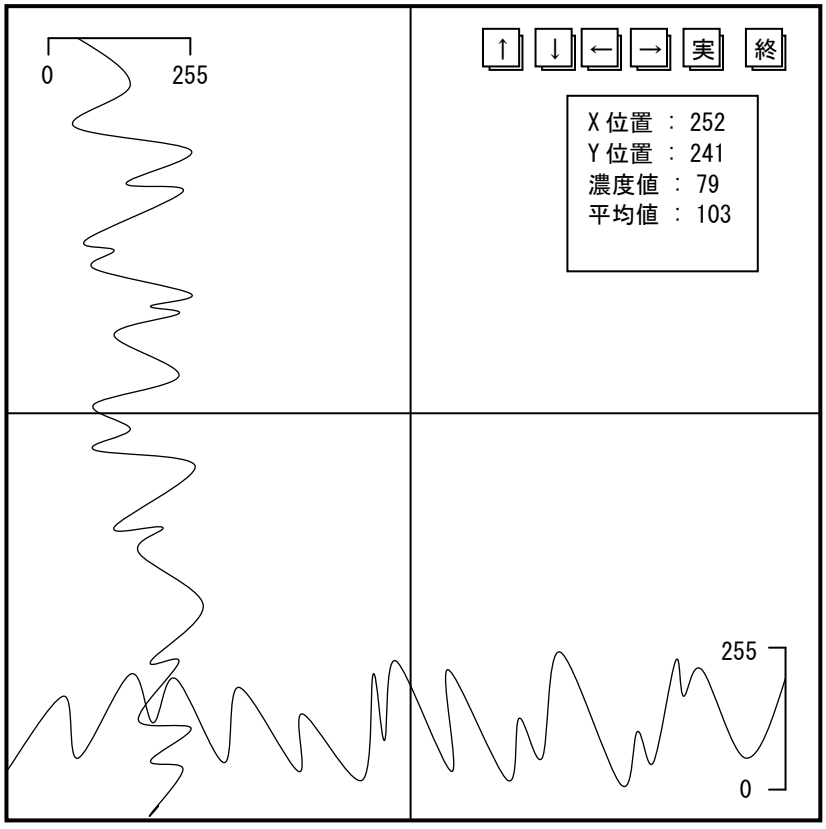
Ｘ断面、Ｙ断面での濃度値をモニタ TV 上に表示するもので、Ｘおよびｙ座標をトラックボール(4 方向キー)を使用して調整します。
ｍ×ｎの拡張画像メモリの 512×480 部分のメモリ状態を表示します。
画像の移動はマウス操作でｘ方向が 128、ｙ方向が 120 ピッチで移動します。

① ***memory_no*** はメモリ番号.です。
　-1(CURRENT_MEMORY)と指定するとカレントメモリとなります。

戻り値

ありません。

操作方法　① モニタ TV 表示



② A スコープ表示内容

X 位置(X pos)：　aaa　　←現在の X 座標位置
Y 位置(Y pos)：　bbb　　←現在の Y 座標位置
濃度値(Level)：　ccc　　←現在の座標位置の濃度値
平均値(Average)：　ddd　　←XY 線下の平均濃度
※ ()は英語モード時の表示内容

③ トラックボール操作

○↑(↑) ･･･････画像の上方向への移動
○↓(↓) ･･･････画像の下方向への移動
○←(←) ･･･････画像の左方向への移動
○→(→) ･･･････画像の右方向への移動
○実(Ex) ･･･････濃度値表示位置の移動
トラックボールを 4 方向に移動･･･････XY 座標位置が 4 方向に移動
(4 方向キー押下)
左ボタンを押す(EXECUTE キー押下) ･･･････操作終了
○終(Qu) ･･･････ 終了

例

Ａスコープを表示します。

```
#include  "f_gui.h"
#include  "f_video.h"
#include  "f_bgray.h"
#include  "f_lsens.h"

void init()
{
    Lib_init_cursor();
    if( NORMAL_RETURN == Lib_lsensor_init
           ( NORMAL_FREEZE, D2048CHN, 2400, 0, 2048, 1024 ))
    {
        Lib_lensor_video_input();
        Lib_ex_ascope( CURRENT_MEMORY );
    }
}
```

留意事項

○ Lib_init_cursor を実行前にコールしてください。
○ Lib_freeze をライブラリ内ではコールしていませんので、メモリ内容と表示内容が異なる場合、結果が異常になります。ご留意ください。
○ Lib_freeze をライブラリ内ではコールしていませんので、メモリ内容と表示内容が異なる場合、結果が異常になります。ご留意ください。
○ 実行時の切り出し表示範囲は Lib_get_extract_window で取得しています。

6. 標準入出力ライブラリ

本ライブラリは 90X シリーズが標準でサービスしている RS232C,C-NET,C-LINK,E-LINK インターフェイスについての高水準ライブラリです。

※FVL/LNX にて RS232C の高速通信を行う場合の注意点

フリーラン中に RS232C にて高速通信(115200bps 等)を行うと通信に失敗する事があります。
その場合は、

・フリーランを停止してから行う
・通信速度を下げて通信を行う

などの対処を行ってください。

Lib_set_io_timeout

機　能

標準入出力のタイムアウト時間発生時間の設定

形　式

```
#include "f_stdio.h"
void Lib_set_io_timeout( unsigned int time );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入出力タイムアウト時間発生時間の設定をします。
デフォルトは 30 秒になっています。

① ***time*** はタイムアウト時間監視時間をミリ秒単位で設定します。

戻り値

ありません。

例

○ タイムアウト時間時間を 5 秒に設定し、入力を待ちます。

```
#include "f_stdio.h"
#define  TIMEOUT  5000

void input()
{
    int ch;
    char s[10];

    Lib_open_io();
    Lib_set_io_timeout( (unsigned int)TIMEOUT );
    if ( EOF == (ch = Lib_getchar( )) )
        Lib_chrdisp( 1, 30, "Time Out(5s)");
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_close_io();
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」, 「RS232C CH1」, 「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

☆ Lib_set_stdin…… 入力対象を更新します。
☆ Lib_get_stdin…… 入力対象を参照します。
☆ Lib_set_stdout…… 出力対象を更新します。
☆ Lib_get_stdout…… 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

Lib_get_io_timeout

機　能

標準入出力のタイムアウト時間発生時間の参照

形　式

```
#include "f_stdio.h"
unsigned int Lib_get_io_timeout( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入出力のタイムアウト時間発生時間を参照します。

戻り値

タイムアウト時間発生時間(ms)を返します。
デフォルトは 30 秒になっています。

例

タイムアウト時間時間を参照し、5 秒でない場合は 5 秒に設定し、入力を待ちます。

```
#include "f_stdio.h"
#define  TIMEOUT  5000

void input()
{
    int ch;
    char s[10];

    Lib_open_io();
    if ( TIMEOUT != Lib_get_io_timeout( ))
        Lib_set_io_timeout( (unsigned int)TIMEOUT );
    if ( EOF == (ch = Lib_getchar( )) )
        Lib_chrdisp( 1, 30, "Time Out(5sec)");
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_close_io();
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin ····· 入力対象を更新します。
- ☆ Lib_get_stdin ····· 入力対象を参照します。
- ☆ Lib_set_stdout ····· 出力対象を更新します。
- ☆ Lib_get_stdout ····· 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

Lib_open_io

機　能

標準入出力の開始

形　式

```
#include "f_stdio.h"
int Lib_open_io( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入出力を開始します。
stdin, stdout 共にオープンします。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

標準入力から入力します。

```
#include "f_stdio.h"

void input()
{
    int ch;
    char s[10];

    Lib_open_io( );
    if ( EOF == (ch = Lib_getchar( )) )
        Lib_chrdisp( 1, 30, "Error!" );
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_close_io( );
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin ····· 入力対象を更新します。
- ☆ Lib_get_stdin ····· 入力対象を参照します。
- ☆ Lib_set_stdout ····· 出力対象を更新します。
- ☆ Lib_get_stdout ····· 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3(システムパラメータ更新/参照)ライブラリの項を参照してください。

Lib_close_io

機　能

標準入出力の終了

形　式

```
#include "f_stdio.h"
int Lib_close_io( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入出力の終了します。
stdin, stdout 共にクローズします。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

標準入力から入力します。

```
#include "f_stdio.h"

void input()
{
    int ch;
    char s[10];

    Lib_open_io( );
    if ( EOF == (ch = Lib_getchar( )) )
        Lib_chrdisp( 1, 30, "Error!" );
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_close_io( );
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin…… 入力対象を更新します。
- ☆ Lib_get_stdin…… 入力対象を参照します。
- ☆ Lib_set_stdout…… 出力対象を更新します。
- ☆ Lib_get_stdout…… 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

Lib_getchar

機　能

標準入力から 1 文字入力

形　式

```
#include "f_stdio.h"
int Lib_getchar( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準 I/O ストリームから 1 文字入力し、int 型で返します。

戻り値

入力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

標準入力から入力します。

```
#include "f_stdio.h"

void input()
{
    int ch;
    char s[10];

    Lib_open_io( );
    if ( EOF == (ch = Lib_getchar( )) )
        Lib_chrdisp( 1, 30, "Error!" );
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_close_io( );
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin …… 入力対象を更新します。
- ☆ Lib_get_stdin …… 入力対象を参照します。
- ☆ Lib_set_stdout …… 出力対象を更新します。
- ☆ Lib_get_stdout …… 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

○ Lib_open_io を実行前にコールする必要があります。

Lib_putchar

機　能

標準出力へ１文字出力

形　式

```
#include "f_stdio.h"
int Lib_putchar( char ch );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準出力へ１文字出力します。

① ***ch*** は出力文字です。

戻り値

正常終了時は出力文字を返します。
エラーが発生すると EOF を返します。

例

標準出力から’A’を出力します。

```
#include "f_stdio.h"

void output()
{

    Lib_open_io( );
    if ( EOF == Lib_putchar( 'A' ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_close_io( );
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin…… 入力対象を更新します
- ☆ Lib_get_stdin…… 入力対象を参照します。
- ☆ Lib_set_stdout…… 出力対象を更新します。
- ☆ Lib_get_stdout…… 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

○ Lib_open_io を実行前にコールする必要があります。

Lib_gets

機　能

標準入力から文字列入力

形　式

```
#include "f_stdio.h"
int Lib_gets( char *string );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入力から文字列を入力します。
文字列の終端はシステムパラメータで設定した「ターミネート」により終了します。

① ***string*** は文字列の先頭を示すポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

標準入力から文字列を入力します。

```
#include "f_stdio.h"

void input()
{
    char string[10];
    char s[30];

    Lib_open_io( );
    if ( NORMAL_RETURN != Lib_gets( string ) )
        Lib_chrdisp( 1, 30, "Error!" );
    else
    {
        Lib_sprintf( s, "Input :%s", string );
        Lib_chrdisp( 1, 30, string );
    }
    Lib_close_io( );
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin‥‥‥‥‥入力対象を更新します。
- ☆ Lib_get_stdin‥‥‥‥‥入力対象を参照します。
- ☆ Lib_set_stdout‥‥‥‥‥出力対象を更新します。
- ☆ Lib_get_stdout‥‥‥‥‥出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

○ Lib_open_io を実行前にコールする必要があります。

FVL/LNX

○ 902 では “Lib_gets()” 関数内部で CR または LF を受信すればターミネートとして判定しておりましたが、FVL/LNX では厳密にシステムパラメータで設定された「ターミネート」を受信しない限り文字列の終端とはみなしません。

Lib_puts

機　能

標準出力から文字列出力

形　式

```
#include "f_stdio.h"
int Lib_puts( char *string );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準出力へ文字列を出力します。
文字列の終端はシステムパラメータで設定した「ターミネート」を出力します。

① ***string*** は文字列の先頭を示すポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

標準出力から 'Test Output' を出力します。

```
#include "f_stdio.h"

void output()
{

    Lib_open_io( );
    if ( NORMAL_RETURN != Lib_puts( "Test Output" ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_close_io( );
}
```

留意事項

○ 標準入出力とは 90X シリーズが標準で用意している
「RS232C CH0」,「RS232C CH1」,「C-NET」,
「C-LINK」,「P-LINK」,「E-LINK」を意味します。
更新/参照するライブラリは以下のようなものが用意されています。

- ☆ Lib_set_stdin…… 入力対象を更新します。
- ☆ Lib_get_stdin…… 入力対象を参照します。
- ☆ Lib_set_stdout…… 出力対象を更新します。
- ☆ Lib_get_stdout…… 出力対象を参照します。

※ 詳細は基本ライブラリ説明書 Vol.3「システムパラメータ更新/参照」ライブラリの項を参照してください。

○ Lib_open_io を実行前にコールする必要があります。

Lib_printf

機　能

書式つき出力

形　式

```
#include "f_stdio.h"
int Lib_printf( char *fmtp, ... );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フォーマットで指定する書式に従って引数の内容を stdout(RS232C, C-NET, C-LINK, P-LINK, E-LINK)へ出力します。

① ***fmtp*** は書式制御文字列へのポインタです。

format の中身の文字はそのまま出力されますが、%で始まる書式制御文字列は、それに対応する引数の書式制御に使用されます。
したがって format で指定した書式制御文字列の型が対応していなかったり、引数の数が不足しているような場合は、出力は保証されません。
ただし、引数の数が書式文字より多い場合には余った引数は単純に捨てられます。

書式制御文字列の一般形式は次のようになります。

% [フラグ] [[0]w] [修飾子]型

% …… 書式制御文字列の開始。

フラグ

- …… フィールドの左端にそろえて出力します。
+ …… 数値の直前に+/-の符号をつけます。
▪ …… 正の数の場合に、符号位置に空白をつけます。
…… o 指定に用いた場合、出力データの前部に 0 をつけます。

☆ x, X 指定に用いた場合、出力データの前部に 0x, 0X をつけます。
☆ f, e, E, G 指定に用いた場合、どんな場合にも小数点をつけて出力します。g, G のときは後続する 0 を切り捨てません。
☆ その他の指定に対しては意味をもちません.

[0]w …… w はフィールド幅を示す整数値です。引数の桁が w の幅より小さければ、左に空白がパッティグされます。引数の桁が w の幅より大きければ w 指定は無視され、引数の桁で出力されます。

.n …… n は小数点部を示す整数値です。引数の小数部が n より大きければまるめられます。

☆ 0 とすると小数点(.)も表示しません。
☆ n を省略すると次のようなディフオルト解釈が行われます。
e, E, f 指定に対しては 6 桁, g, G 指定に対してはすべての桁になります。
☆ s に対する指定では出力する最大文字数の指定になります。
☆ d, i, u, o, x, X に対する指定では出力する最小桁数の指定になります。

修飾子 …… <型>を修飾するもので次の 3 つがあります。

H …… d, i, o, u, x, X に対し short int を指定します。
L …… d, i, o, u, x, X に対し long int を、e, E, f, g, G に対し double を指定します。
L …… e, E, f, g, G に対し long double を指定します。

型 …… 引数の型を指定します。

D …… 符号つき 10 進整数
i …… 符号つき 10 進整数
o …… 符号なし 8 進整数
u …… 符号なし 10 進整数
x …… 符号なし 16 進整数(小文字表示。1f0a)
X …… 符号なし 16 進整数(大文字表示。1F0A)
F …… 浮動小数点(10 桁)
e …… 指数形式(小文字表示。1.2e+003)
E …… 指数形式(大文字表示。1.2E+003)
C …… 1 文字
s …… 文字列
n …… %がくるまでに出力した文字数を対応する引数に格納します。

戻り値

処理文字数を返します。

例

書式例

```
#include "f_stdio.h"

int ptest( )
{
    int val=255;
    Lib_printf( "%d %o %x",val, val, val );
}
```

留意事項

ありません。

FVL/LNX

○ 標準出力に「E-LINK」を選択した場合は、Lib_open_io を実行前にコールする必要があります。

Lib_sprintf

機　能

文字列への書式つき出力

形　式

```
#include "f_stdio.h"
int Lib_sprintf( char *string,  char *fmtp,  ... );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字列 string に、フォーマットで指定する書式に従って引数の内容を書き出します。
書式制御は Lib_printf と同じです。

戻り値

処理文字数を返します。

例

書式例

```
#include "f_stdio.h"

int ptest( )
{
    int val=255;
    char string[30];
    Lib_sprintf( string,"%d %o %x",val, val, val );
    Lib_chrdisp( 1, 30, string );
}
```

留意事項

ありません。

Lib_scanf

機　能

書式つき入力

形　式

```
#include "f_stdio.h"
int  Lib_scanf( char *fmtp, ... );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

フォーマットで指定する書式に従って引数の内容を stdin から入力します。

① ***fmtp*** は書式制御文字列へのポインタです。

format の中身の文字はそのまま出力されますが、%で始まる書式制御文字列は、それに対応する引数の書式制御に使用されます。
したがって format で指定した書式制御文字列の型が対応していなかったり、引数の数が不足しているような場合は、出力は保証されません。
ただし、引数の数が書式文字より多い場合には余った引数は単純に捨てられます。

書式制御文字列の一般形式は次のようになります。

% [フラグ] [[0]w] [修飾子]型

% …… 書式制御文字列の開始。

フラグ

- …… フィールドの左端にそろえて入力します。
+ …… 数値の直前に+/-の符号をつけます。
▪ …… 正の数の場合に、符号位置に空白をつけます。
…… o 指定に用いた場合、入力データの前部に 0 をつけます。

☆ x, X 指定に用いた場合、入力データの前部に 0x, 0X をつけます。
☆ f, e, E, G 指定に用いた場合、どんな場合にも小数点をつけて入力します。g, G のときは後続する 0 を切り捨てません。
☆ その他の指定に対しては意味をもちません.

[0]w …… w はフィールド幅を示す整数値です。引数の桁が w の幅より小さければ、左に空白がパッティグされます。
引数の桁が w の幅より大きければ w 指定は無視され、引数の桁で入力されます。

.n …… n は小数点部を示す整数値です。引数の小数部が n より大きければまるめられます。

☆ 0 とすると小数点(.)も表示しません。
☆ n を省略すると次のようなディフオルト解釈が行われます。
e, E, f 指定に対しては 6 桁,g, G 指定に対してはすべての桁になります。
☆ s に対する指定では入力する最大文字数の指定になります。
☆ d, i, u, o, x, X に対する指定では入力する最小桁数の指定になります。

修飾子 …… <型>を修飾するもので次の 2 つがあります。

h …… d, i, o, u, x, X に対し short int を指定します。
l …… d, i, o, u, x, X に対し long int を指定します。

型 …… 引数の型を指定します。

d …… 符号つき 10 進整数
i …… 符号つき 10 進整数
o …… 符号なし 8 進整数
u …… 符号なし 10 進整数
x …… 符号なし 16 進整数 (小文字表示。1f0a)
X …… 符号なし 16 進整数 (大文字表示。1F0A)
c …… 1 文字
s …… 文字列
n …… %がくるまでに入力した文字数を対応する引数に格納します。

戻り値

処理文字数を返します。

例

書式例

```
#include "f_stdio.h"

int smpl_scanf()
{
    int val1,val2;
    Lib_open_io();
    Lib_printf("¥n¥r?Input = ");
    Lib_scanf("%d %d",&val1,&val2);
    Lib_dispval(1,28,val1);
    Lib_dispval(1,29,val2);
    Lib_close_io();
}
```

留意事項

ありません。

FVL/LNX

○ 標準入力に「E-LINK」を選択した場合は、Lib_open_io を実行前にコールする必要があります。

Lib_sscanf

機　能

文字列からの書式つき入力

形　式

```
#include "f_stdio.h"
int  Lib_sscanf( char *string,  char *fmtp, ... );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

文字配列 string からフォーマットで指定する書式に従って引数の内容を書き込みます。
書式制御は Lib_scanf と同じです。

戻り値

処理文字数を返します。

例

書式例

```
#include "f_stdio.h"

int ptest( )
{
    int  val;
    char *str,string[30];

    str = "Value 1234";
    Lib_sscanf( str, "%s%d",string, &val );
    Lib_chrdisp( 1, 29, string );
    Lib_ddeciml( 6, 30, val );
}
```

留意事項

ありません。

Lib_get_io_error

機　能

標準入出力のエラー情報参照

形　式

```
#include "f_stdio.h"
int Lib_get_io_error( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入出力でのエラー情報を参照します。

戻り値

C-NET,C-LINK,RS232C の各ライブラリ説明書を参照してください。

例

```
#include "f_stdio.h"

void put( char ch )
{
    char string[20];

    if( EOF == Lib_putchar( ch ) )
    {
        Lib_sprintf( string, "ERROR CODE:%d", Lib_get_io_error() );
        Lib_chrdisp( 1, 1, string );
    }
}
```

留意事項

ありません。

Lib_io_buff_check

機　能

標準入出力用バッファの入力チェック

形　式

```
#include "f_stdio.h"
int Lib_io_buff_check( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

標準入出力用バッファの入力チェックを行います。

戻り値

値	定　　数	意　　味
0	NOT_INPUT_BUFFER	バッファに入力していない。
3	なし	バッファに入力している。

例

C-NET を使用し、バッファに入力している場合引き取る。

```
#include "f_stdio.h"
#include "f_graph.h"

void get()
{
    char string[20];

    if ( NOT_INPUT_BUFFER != Lib_io_buff_check() )
    {
        if ( NORMAL_RETURN == Lib_gets( string ))
            Lib_chrdisp( 1, 30, string );
    }
}
```

留意事項

ありません。

7. RS232C ライブラリ

本ライブラリは 90X シリーズに標準でサービスしている RS232C の入出力に関するものです。

※FVL/LNX にてRS232C の高速通信を行う場合の注意点

フリーラン中にRS232C にて高速通信(115200bps 等)を行うと通信に失敗する事があります。
その場合は、

・フリーランを停止してから通信を行う
・通信速度を下げて通信を行う

などの対処を行ってください。

Lib_opensio

機　能

RS232C 入出力オープン

形　式

```
#include "f_stdio.h"
int Lib_opensio( int channel );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定チャネルの RS232C のコントローラを初期化(システムパラメータで設定された各パラメータを使用)を実施し、入出力可能状態とします。
このときバッファはクリアされます。

① ***channel*** はチャネル No. です。

値	定　　数	意　　味
0	SIO_CHANNEL0	チャネル No. 0 を指定します。
1	SIO_CHANNEL1	チャネル No. 1 を指定します。

初期化に際し、参照しているシステムパラメータは以下の通りです。

☆ビット･レート
☆データ･ビット長
☆ストップ･ビット長
☆パリティ･ビット長
☆フロー制御

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

8 バイトをチャネル No.0 から入力します。

```
#include "f_stdio.h"

void set_baudrate()
{
    int inbyte;
    char buff[10];

    Lib_opensio( SIO_CHANNEL0 );
    if ( NORMAL_RETURN != Lib_getsi( SIO_CHANNEL0, WAITING, 8, buff, &inbyte ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_chrdisp( 1, 30, buff );
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

○ オープンしていないチャネルについての入出力および、クローズはできません。

Lib_speed

機　能

RS232C の設定

形　式

```
#include "f_stdio.h"
int Lib_speed( int channel, int baud_rate,
                    int data_bit, int stop_bit, int parity, int x_parameter );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

RS232C の各パラメータの設定をします。このときバッファはクリアされます。

① ***channel*** はチャネル No. です。

値	定　　数	意　　味
0	SIO_CHANNEL0	チャネル No. 0 を指定します。
1	SIO_CHANNEL1	チャネル No. 1 を指定します。

② ***baud_rate*** はボーレートです。

値	定　　数	意　　味
1200	RATE1200BPS	1200bps に設定します。
1800	RATE1800BPS	1800bps に設定します。
2400	RATE2400BPS	2400bps に設定します。
4800	RATE4800BPS	4800bps に設定します。
9600	RATE9600BPS	9600bps に設定します。
19200	RATE19200BPS	19200bps に設定します。
38400	RATE38400BPS	38400bps に設定します。
57600	RATE57600BPS	57600bps に設定します。
76800	RATE76800BPS	76800bps に設定します。
115200	RATE115200BPS	115200bps に設定します。

③ ***data_bit*** はデータ･ビット長です。

値	定　　数	意　　味
5	DATA_BIT_5	5 ビットに設定します。
6	DATA_BIT_6	6 ビットに設定します。
7	DATA_BIT_7	7 ビットに設定します。
8	DATA_BIT_8	8 ビットに設定します。

④ ***stop_bit*** はストップ･ビット長です。

値	定　　数	意　　味
0	STOP_BIT_1	1 ビットに設定します。
1	STOP_BIT_1_5	1.5 ビットに設定します。
2	STOP_BIT_2	2 ビットに設定します。

⑤ ***parity*** はパリティ･ビット長

値	定　数	意　味
0	NONE	なしに設定します。
1	ODD	奇数に設定します。
2	EVEN	偶数に設定します。

⑥ ***x_parameter*** は Xon/Xoff 制御です。

値	定　数	意　味
0	FLOW_NONE	フロー制御なしに設定します。
1	FLOW_XONOFF	Xon/Xoff フロー制御有効に設定します。
2	FLOW_RSCS	Rs/Cs フロー制御有効に設定します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

ビット･レートを変更します。

```
#include "f_stdio.h"
#define  BAUDRATE     2400
#define  DATABIT      5
#define  STOPBIT      2

void set_baudrate()
{
    int inbyte;
    char buff[10];

    Lib_opensio( SIO_CHANNEL0 );
    if( NORMAL_RETURN != Lib_speed( SIO_CHANNEL0 , BAUDRATE ,
                                    DATABIT , STOPBIT , ODD , FLOW_RSCS ))

    if ( NORMAL_RETURN != Lib_getsi( SIO_CHANNEL0, WAITING, 8, buff, &inbyte ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

○ このライブラリを実行するときは必ず先に前項の Lib_opensio をコールしてください。
○ 901NT, 902 では、ストップ･ビット長に 1.5 ビットの設定はできません。
○ ボーレート設定範囲は以下の通りです。

901	1200～19200
902	1200～57600,115200
	(38400 以上に関しては弊社標準ケーブルでのみサポートします。)
903	1200～76800
904	1200～76800
FVL/LNX	1200～57600,115200(※FVL/LNX 留意事項を参照)

〇フリーラン中に RS232C にて高速通信(115200bps 等)を行うと通信に失敗する事があります。
その場合は、

・フリーランを停止してから行う
・通信速度を下げて通信を行う

などの対処を行ってください。

Lib_getsi

機　能

RS232C データ入力

形　式

```
#include "f_stdio.h"
int Lib_getsi( int channel, int option, unsigned char req_byte,
                                        char *inbuff, int *in_byte );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

RS232C 指定チャネルからのデータを引取るもので、1～128 バイトまでのデータを一度に引取ることができます。

① ***channel*** はチャネル No. です。

値	定　数	意　味
0	SIO_CHANNEL0	チャネル No.0 を指定します。
1	SIO_CHANNEL1	チャネル No.1 を指定します。

② ***option*** は入力完了待ちオプションです。

値	定　数	意　味
0	NO_WAIT	バッファが空になった場合は終了します。現在受信バッファに滞納しているデータを入力します。(受信バッファが空ならそのままリターン)
1	WAITING	入力要求バイトになるまで入力します。 但し、入出力タイムアウト時間監視時間経過しても満たされない時または、何らかのハードウェア異常検出時には、エラーリターンする。

③ ***req_byte*** は入力要求バイト数(1～128)です。

④ ****inbuff*** は入力データバッファのポインタです。

⑤ ****in_byte*** は入力バイト数を出力します。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
4	HARDWARE_ERR	ハードウェア異常が発生しました。
5	TIMEOUT_ERR	タイムアウト時間が発生しました。
-1	ERROR_RETURN	パラメータエラーです。

例

8 バイトをチャネル No.0 から入力します。

```
#include "f_stdio.h"

void set_baudrate()
{
    int inbyte;
    char buff[10];

    Lib_opensio( SIO_CHANNEL0 );
    if ( NORMAL_RETURN != Lib_getsi( SIO_CHANNEL0, WAITING, 8, buff, &inbyte ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_chrdisp( 1, 30, buff );
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

○ 標準入出力タイムアウト時間はデフォルトで 30 秒になっています。
　設定する場合は Lib_set_io_timeout を実行してください。
○ Lib_opensio を実行前にコールする必要があります。

Lib_putso

機　能

RS232C データ出力

形　式

```
#include "f_stdio.h"
int Lib_putso( int channel, int req_byte, char *outbuff );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

RS232C 指定チャネルへデータを送出するもので、1～128 バイトまでのデータを一度に送出することができます。

① ***channel*** はチャネル No. です。

値	定　　数	意　　味
0	SIO_CHANNEL0	チャネル No.0 を指定します。
1	SIO_CHANNEL1	チャネル No.1 を指定します。

② ***req_byte*** は出力要求バイト数(1～128)です。

③ ****outbuff*** は出力データバッファのポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
5	TIMEOUT_ERR	タイムアウト時間が発生しました。
-1	ERROR_RETURN	パラメータエラーです。

例

"Test Output" をチャネル No.0 から出力します。

```
#include "f_stdio.h"

void set_baudrate()
{
    Lib_opensio( SIO_CHANNEL0 );
    if ( NORMAL_RETURN != Lib_putso( SIO_CHANNEL0, 10, "Test Output" ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

○ タイムアウト時間はデフォルトで 30 秒になっています。
　設定する場合は Lib_set_io_timeout を実行してください。
○ Lib_opensio を実行前にコールする必要があります。

Lib_getc_rs232c

機　能

RS232C 1 文字入力

形　式

```
#include "f_stdio.h"
int Lib_getc_rs232c( int channel );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

RS232C の指定チャネル No. から 1 文字入力し、int 型で返します。

① ***channel*** はチャネル No. です。

値	定　　数	意　　味
0	SIO_CHANNEL0	チャネル No. 0 を指定します。
1	SIO_CHANNEL1	チャネル No. 1 を指定します。

戻り値

入力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

チャネル No. 0 から入力します。

```
#include "f_stdio.h"

void input()
{
    int ch;
    char s[10];

    Lib_opensio( SIO_CHANNEL0 );
    if ( EOF == (ch = Lib_getc_rs232c(SIO_CHANNEL0 )) )
        Lib_chrdisp( 1, 30, "Error!" );
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

○ タイムアウト時間はデフォルトで 30 秒になっています。
　設定する場合は Lib_set_io_timeout を実行してください。
○ Lib_opensio を実行前にコールする必要があります。

Lib_putc_rs232c

機　能

RS232C 1 文字出力

形　式

```
#include "f_stdio.h"
int Lib_putc_rs232c( char ch, int channel );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

RS232C の指定チャネル No. から 1 文字出力します。

① ***ch*** は出力文字です。

② ***channel*** はチャネル No. です。

値	定　数	意　味
0	SIO_CHANNEL0	チャネル No. 0 を指定します。
1	SIO_CHANNEL1	チャネル No. 1 を指定します。

戻り値

正常終了時は出力文字を返します。
エラーが発生すると EOF を返します。

例

チャネル NO. 0 から’A’を出力します。

```
#include "f_stdio.h"

void output()
{

    Lib_opensio( SIO_CHANNEL0 );
    if ( EOF == Lib_putc_rs232c( 'A',SIO_CHANNEL0 ) )
    {
        Lib_chrdisp( 1, 30, "Error!" );
    }
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

○ タイムアウト時間はデフォルトで 30 秒になっています。
　設定する場合は Lib_set_io_timeout を実行してください。
○ Lib_opensio を実行前にコールする必要があります。

Lib_clossio

機　能

RS232C 入出力クローズ

形　式

```
#include "f_stdio.h"
int  Lib_clossio( int channel );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

シリアル入出力をクローズします。

① ***channel*** はチャネル No.です。

値	定　　数	意　　味
0	SIO_CHANNEL0	チャネル No.0 を指定します。
1	SIO_CHANNEL1	チャネル No.1 を指定します。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

チャネル No.0 から入力します。

```
#include "f_stdio.h"

void input()
{
    int ch;
    char s[10];

    Lib_opensio( SIO_CHANNEL0 );
    if ( EOF == (ch = Lib_getc_rs232c(SIO_CHANNEL0 )) )
        Lib_chrdisp( 1, 30, "Error!" );
    else
    {
        Lib_sprintf( s, "Input :%c", ( ch & 0xff ) );
        Lib_chrdisp( 1, 30, s );
    }
    Lib_clossio( SIO_CHANNEL0 );
}
```

留意事項

ありません。

8. C-NET 基本ライブラリ

「基本ライブラリ」には C-NET を使用するために最低限必要なライブラリについて、記載されています。
通常は「C-NET 応用ライブラリ」を御使用ください。
各ライブラリ関数に対しては、詳細な解説、形式、インクルードファイル、プロトタイプ、引き数、戻り値が説明されています。

Lib_open_cnet

機　能

C-NET オープン

形　式

```
#include "f_cnet.h"
int Lib_open_cnet( int node_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

C-NET のノード番号を指定し、初期化を行います。
初期化後、指定したノードが存在しているのを確認後終了します。
この初期化の後で、接続の解除を行わないで、再び初期化した場合は、自動的に接続の解除が行われてから初期化します。

① ***node_no*** は、自局のノード番号(8～63)です。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了
6	LINE_BUSY	伝送路が 290msec 以上ビジィ状態であったため、初期化を中断し設定を行わなかった。
-1	ARG_ERR	パラメータの値が 8～63 の範囲内にない。
-3	ALREADY_EXIST	既にノードが存在している。
888	HW_ERROR	ハードウェア障害が発生した。
999	UN_KNOWN	不明の障害が発生した。

例

プロントを送信後、コマンドＧの入力を待ちメッセージを出力します。

```
#include "f_pinf.h"
#include "f_cnet.h"
#include "f_stdlib.h"
#include "f_graph.h"

#define  MAX_LENGTH      2048
#define  MAX_CHRDISP     64

static unsigned char put_buff[30] = "C-NET のテスト送信";

void cnet()
{
    int node_no;
    int status;
    int ch;
    int len, f_type;
    unsigned char *buff;

    buff = (unsigned char *)Lib_mlalloc( MAX_LENGTH );
    node_no = Lib_get_cnet_node_no();
    if( NO_ERROR == Lib_open_cnet( node_no ))
    {
        for( status = OFF; ON != status ; )
        {
            if( EOF != Lib_putc_cnet( PROMPT, NODE_0 ))
            {
                if( EOF != (ch = Lib_getc_cnet( NODE_0 )))
                {
                    if( 'G' == (ch & 0xff) )
                        status = ON;
                }
            }
        }
        if( NO_ERROR == Lib_puts_cnet( put_buff,
                strlen( (char *) put_buff ), TYPE_COMMAND, NODE_0  ))
        {
            if ( NO_ERROR == Lib_gets_cnet( buff, &len, &f_type, NODE_0 ))
            {
                if(( MAX_CHRDISP >= len ) && TYPE_COMMAND == f_type ))
                Lib_chrdisp( 1, 1, (char *)buff );
            }
        }
    }
    Lib_lfree( (char *) buff );
    Lib_close_cnet(  );
}
```

留意事項

ありません。

Lib_close_cnet

機　能

C-NET クローズ

形　式

```
#include "f_cnet.h"
int Lib_close_cnet( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

C-NET 接続を解除します。
C-NET 接続を解除した場合、再び初期化されるまで、C-NET は利用できません。

戻り値

値	定　　数	意　　味
0	NO_ERROR	正常終了しました。
-2	NO_CONNECT	C-NET の初期化がされていない。 (接続されていない)

例

Lib_open_cnet 関数の例を参照してください。

留意事項

ありません。

Lib_cnet_irq_set

機　能

受信割り込み処理関数の登録

形　式

```
#include "f_cnet.h"
int Lib_cnet_irq_set(void (*service_proc)());
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信処理を C-NET インターフェイスからの受信割り込みで実行するような場合、この制御関数によって、あらかじめ処理関数を登録しておく必要があります。
登録は C-NET インターフェイスの初期化後に行ってください。
受信処理関数内で使用可能な C-NET ライブラリは、Lib_cnet_receive 関数のみです。
また、割り込まれる側のルーチンでは以下の C-NET ライブラリは使用できません。

- ☆ Lib_cnet_receive_status
- ☆ Lib_cnet_receive
- ☆ Lib_gets_cnet
- ☆ Lib_getc_cnet

① ***service_proc*** は割り込み処理関数のポインタです。
初期化の際には NULL になっています。NULL の場合は何もしません。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	引数の値が異常です。
-2	NO_CONNECT	まだ接続されていません。

例

割り込みで受信処理を行い、受信データをバッファに格納します。

```
#include "f_cnet.h"

static unsigned char message[128];    /* 受信データパケット */
static unsigned char data[2048];      /* 受信データパケット */
static char buff[ 100 ];              /* 受信バッファ */
static int buffp;                     /* 受信バッファ インデックス */

void rcv_int( void );

void main( void )
{
    if ( NO_ERROR == Lib_open_cnet( NODE_8 ) )
    {
        buffp = 0;                    /* 受信バッファ インデックスのクリア */
        Lib_cnet_irq_set( rcv_int );  /* 割り込み処理関数の登録 */

        /* 何らかの処理が入ります。*/

        Lib_cnet_irq_clear();         /* 割り込み処理関数の登録解除 */
        Lib_close_cnet();             /* C-NET の終了 */
    }
}

/* C-NET 受信割り込み処理*/
void rcv_int( void )
{
    int node;/* 送信元ノード番号 */
    int len;/* 受信データ長 */
    int rcv_len;/* 受信有効長 */
    int i;

    if ( NO_ERROR == Lib_cnet_receive( message, data, &node, &len ) )
    {
        rcv_len = *(long *)message[12];/* 受信有効長 */
        /* 受信データをバッファに格納 */
        for ( i = 0; i < rcv_len; i++ )
        {
            buff[ buffp++ ] = data[ i ];
            if ( buffp == 100 )
                buffp = 0;
        }
    }
}
```

留意事項

○ 後述する受信バッファ処理関数は、本関数を使用し C-NET ライブラリ内の割り込み処理関数を登録しています。

○ 割り込みでは使用できる C ライブラリ関数に制限があります。
まず、マウス、キー入力関数、シリアル入出力関数、タイマ関数はすべて使用できません。
また、リエントラント(再入可能)でない関数も使用できません。

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnet_irq_clear

機　能

受信割り込み処理関数の登録解除

形　式

```
#include "f_cnet.h"
int Lib_cnet_irq_clear(void);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信処理を C-NET インターフェイスからの受信割り込みで実行しているような場合、この制御関数によって、その動作を停止できます。
再起動する場合は、再度、処理関数の登録を行う必要があります。

戻り値

値	定　　数	意　　味
0	NO_ERROR	正常終了しました。
-2	NO_CONNECT	まだ接続されていません。

例

Lib_cnet_irq_set を参照してください。

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnet_get_node_map

機　能

C-NET 上のノード・マップの取得

形　式

```
#include "f_cnet.h"
int Lib_cnet_get_nodemap(unsigned char *node_map);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

C-NET 上に存在するノードを調べて報告します。
戻り値(6)は異常ではありませんので、リトライ可能です。

① ***node_map*** は、ノード･マップ格納用配列のポインタです。
配列は、unsigned char 型で 64 要素確保してください。
［例］unsigned char node_map[64];
ノード番号をインデックスとして、配列内容が 0xFF の場合は該当ノードが存在し、0x00 の場合は存在しません。

戻り値

値	定　　数	意　　味
0	NO_ERROR	正常終了しました。
6	LINE_BUSY	伝送路が 290msec 以上ビジィ状態であったため中断しました。
-2	NO_CONNECT	まだ接続されていません。
888	HW_ERROR	インターフェイスまたは制御関数に異常が発生しました。
999	UN_KNOWN	インターフェイスまたは制御関数に未知の異常が発生しました。

例

ノードマップを取得し、存在するノードを表示します。

```
#include "f_graph.h"
#include "f_cnet.h"

void main( void )
{
    unsigned char node_map[ 64 ];
    int i;

    if ( NO_ERROR == Lib_open_cnet( NODE_0 ) )
    {
        Lib_cnet_get_node_map( node_map );
        for ( i = 0; i < 64; i++ )
        {
            if ( node_map[ i ] == 0xff )
                Lib_chrdisp(1+i,10,"x");
        }
        Lib_close_cnet();
    }
}
```

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnet_receive_status

機　能

受信ステータスの取得

形　式

```
#include "f_cnet.h"
int Lib_cnet_receive_status(void);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信の状態を調べて報告します。

戻り値

値	定　　数	意　　味
0	COMPLETE	受信完了しました。受信処理を行う必要があります。
3	READY	受信待機中です。
-2	NO_CONNECT	まだ接続されていません。
888	HW_ERROR	インターフェイスまたは制御関数に異常が発生しました。
999	UN_KNOWN	インターフェイスまたは制御関数に未知の異常が発生しました。

例

受信完了後、受信処理を行います。

```
#include "f_cnet.h"

static unsigned char message[ 128 ];
static unsigned char data[ 2048 ];

void main( void )
{
    int node;
    int length;

    if ( NO_ERROR == Lib_open_cnet( NODE_8 ) )
    {
        while ( COMPLETE != Lib_cnet_receive_status() )
        {
            /* 何らかの処理 */
        }
        Lib_cnet_receive( message, data, &node, &length );

        Lib_close_cnet();
    }
}
```

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnet_receive

機　能

受信処理

形　式

```
#include "f_cnet.h"
int Lib_cnet_receive(unsigned char *message,
                     unsigned char *data, int *node, int *length);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信したデータパケットのメッセージフィールドとデータフィールドを引き取るための制御関数です。

① ***message*** は受信メッセージフィールドを格納する配列のポインタです。
配列は、unsigned char 型で 128 要素必要です。
[例]unsigned char message[128];

② ***data*** は受信データフィールドを格納する配列のポインタです。
配列は、unsigned char 型で 2048 要素必要です。
[例]unsigned char data[2048];

③ ***node*** は送信先ノード番号を格納する変数のポインタです。

値	定　数	意　味
0	NODE_0	ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	ノード番号 63 を指定します。

④ ***length*** は受信データ長を格納する変数のポインタです。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
4	RX_NOTHING	まだ受信していません。 受信待機中です。
-2	NO_CONNECT	まだ接続されていません。

例

Lib_cnet_receive_status を参照してください。

留意事項

○ メッセージフィールドとデータフィルドの構造については、9.2　データパケット構造を参照してください。
○ Lib_open_cnet を実行前にコールする必要があります。
○ ホストと 90X シリーズ間のファイルの受信は「ファイルシステムライブラリ」の Lib_fcopy を使用すると容易です。

Lib_cnet_send_status

機　能

送信ステータスの取得

形　式

```
#include "f_cnet.h"
int Lib_cnet_send_status(void);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

送信の状態を調べて報告します。
戻り値(1,2,6)は異常ではありませんので、リトライ可能です。

戻り値

値	定　数	意　味
0	NO_ERROR	送信完了しました。
1	NOT_READY	送信先のノードが受信セットアップされていませんでした。
2	NO_ACK	受信先のノードからアクノレッジが返ってこないので中止しました。
5	CNET_TRANSMIT	送信中です。
6	LINE_BUSY	伝送路が 290msec 以上ビジィ状態であったため中断しました。
-2	NO_CONNECT	まだ接続されていません。
888	HW_ERROR	インターフェイスまたは制御関数に異常が発生しました。
999	UN_KNOWN	インターフェイスまたは制御関数に未知の異常が発生しました。

例

一文字”A”を送信します。

```
#include "f_cnet.h"

static unsigned char message[ 128 ];
static unsigned char data[ 2048 ];

void main( void )
{
    if ( NO_ERROR == Lib_open_cnet( NODE_8 ) )
    {
        memset( message, 0x1a, 128 );    /* C-NETバッファの初期化 */
        memset( data, 0x1a, 2048 );   /* C-NETバッファの初期化 */

        *(short *)( &message[0]  ) = 0x0001;     /* モード */
        *(short *)( &message[10] ) = 0x0001;     /* データタイプ */
        *(long  *)( &message[12] ) = 1;          /* データ有効長 */
        message[16]                = 0x01;       /* タイプ */
        data[0] = 'A';
        if ( NO_ERROR ==Lib_cnet_send( 8, 16, message, data ) )
        {
            while (CNET_TRANSMIT == Lib_cnet_send_status() )
            {
                /* 何らかの処理 */
            }
        }
        Lib_close_cnet();
    }
}
```

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnet_send

機　能

送信処理

形　式

```
#include "f_cnet.h"
int Lib_cnet_send(int node, int length,
                  unsigned char *message, unsigned char *data);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

メッセージとデータをパケットにして送信するための制御関数です。

① ***node*** 送信先のノード番号です。

値	定　数	意　味
0	NODE_0	ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	ノード番号 63 を指定します。

② ***length*** は送信データ長(16～2048)です。
送信データ長の最小値は 16(バイト)です。最小値に満たない場合も 16 で指定します。

③ ****message*** は送信メッセージフィールド配列のポインタです。
配列は unsigned char 型で 128 要素必要です。
[例]unsigned char message[128];

④ ****data*** は送信データフィールド配列のポインタです。
配列は unsigned char 型で 2048 要素必要です。
[例]unsigned char data[2048];

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	引数の値が異常です。
-2	NO_CONNECT	まだ接続されていません。
-4	NOT_COMPLETE	まだ送信が完了していません。
-5	SEND_TO_ME	送信先が自局になっています。

例

Lib_cnet_send_status を参照してください。

留意事項

○ メッセージフィールドとデータフィールドの構造については、9.2　データパケット構造を参照してください。
○ Lib_open_cnet を実行前にコールする必要があります。
○ ホストと 90X シリーズ間のファイルの送信は「ファイルシステムライブラリ」の Lib_fcopy を使用すると容易です。

9. C-NET 応用ライブラリ

「C-NET 応用ライブラリ」には「基本ライブラリ」を使用した、よりユーザーに近いレベルのライブラリについて記載されています。

各ライブラリ関数に対しては、詳細な解説、形式、インクルードファイル、プロトタイプ、引き数、戻り値が説明されています。

Lib_gets_cnet

機　能

C-NET 文字列受信

形　式

```
#include "f_cnet.h"
int Lib_gets_cnet( unsigned char *data, int *length,int *f_type, int node_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

指定ノードからデータを受信します。
指定ノード以外のデータを受信した場合は破棄します。

① ***data*** は、受信データ格納用バッファへのポインタです。

② ***length*** は、受信データ長(1～2048)を返します。

③ ***f_type*** は、受信ファイルタイプを返します。

値	定　数	意　味
0	TEXT_TYPE	テキスト･タイプのファイルを受信しました。
1	BINARY_TYPE	バイナリ･タイプのファイルを受信しました。

④ ***node_no*** は、受信先のノード番号です。

値	定　数	意　味
0	NODE_0	受信先のノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先のノード番号 63 を指定します。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
4	RX_NOTHING	受信を完了していない。
-2	NO_CONNECT	C-NET BIOS の初期化がされていない。(接続していない)
888	HW_ERROR	ハードウェア障害が発生した。
999	UN_KNOWN	不明の障害が発生した。
9999	IO_TIMEOUT	I/O タイムアウト時間エラーが発生した。

例

Lib_open_cnet 関数の例を参照してください。

留意事項

○ 受信データ格納用バッファの容量は 2048 バイト分必要です。
○ Lib_open_cnet を実行前にコールする必要があります。
○ ホストと 90X シリーズ間のファイルの受信は「ファイルシステムライブラリ」の Lib_fcopy を使用すると容易です。

Lib_getc_cnet

機　能

C-NET 1 文字受信

形　式

```
#include "f_cnet.h"
int Lib_getc_cnet( int node_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

1 バイトを指定のノード番号から入力します。
指定ノード以外のデータを受信した場合は破棄します。

① ***node_no*** は、受信先のノード番号です。

値	定　数	意　味
0	NODE_0	受信先のノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先のノード番号 63 を指定します。

戻り値

入力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

Lib_open_cnet 関数の例を参照してください。

留意事項

○ エラー内容を参照したい場合は、Lib_get_io_error 関数をコールしてください。
○ Lib_open_cnet を実行前にコールする必要があります。

Lib_puts_cnet

機　能

C-NET 文字列送信

形　式

```
#include "f_cnet.h"
int Lib_puts_cnet( unsigned char *data, int length,int f_type, int node_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

C-NET が指定されたノード番号にデータを送信します。
メッセージ･フィールドに規定値をセット後送信しています。

① ***data*** は、送信データバッファへのポインタです。

② ***length*** は、送信データ長(1～2048)です。

③ ***f_type*** は、送信ファイルタイプです。

値	定　　数	意　　味
0	TEXT_TYPE	テキスト･タイプのファイルを送信します。
1	BINARY_TYPE	バイナリ･タイプのファイルを送信します。

④ ***node_no*** は、送信先のノード番号です。

値	定　　数	意　　味
0	NODE_0	受信先のノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先のノード番号 63 を指定します。

戻り値

値	定　　数	意　　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	パラメータの値が異常である。
-2	NO_CONNECT	C-NET BIOS の初期化がされていない。(接続していない)
-4	NOT_COMPLETE	先の送信が完了していない。
-5	SEND_TO_ME	ノード番号が自局の番号と同一である。
888	HW_ERROR	ハードウェア障害が発生した。
999	UN_KNOWN	不明の障害が発生した。
9999	IO_TIMEOUT	I/O タイムアウト時間エラーが発生した。

例

Lib_open_cnet 関数の例を参照してください。

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。
○ ホストと 90X シリーズ間のファイルの受信は「ファイルシステムライブラリ」の Lib_fcopy を使用すると容易です。

Lib_putc_cnet

機　能

C-NET 1 文字送信

形　式

```
#include "f_cnet.h"
int Lib_putc_cnet( char ch, int node_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

1 バイトを指定のノード番号へ送信します。
メッセージ･フィールドに規定値をセット後送信しています。

① ***ch*** は、出力文字です。

② ***node_no*** は、送信先のノード番号です。

値	定　数	意　味
0	NODE_0	受信先のノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先のノード番号 63 を指定します。

戻り値

出力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

Lib_open_cnet 関数の例を参照してください。

留意事項

○ エラー内容を参照したい場合は、Lib_get_io_error 関数をコールしてください。
○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnetbuffopen

機　能

受信バッファ処理の開始

形　式

```
＃include "f_cnet.h"
void Lib_cnetbuffopen( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信バッファ処理は、割り込みで受信処理を行い指定した受信バッファに受信データを格納します。
受信バッファ処理を行う場合は、C-NET インターフェイスを初期化後、この関数をコールする必要があります。
受信バッファ処理に関連する関数は、

```
Lib_cnetbuffopen
Lib_cnetbuffclose
Lib_setcnetrcv
Lib_clrcnetrcv
Lib_getcnetrcv
Lib_cnetbuffcheck
Lib_cnetbuffrcv
Lib_cnetbuffclr
Lib_cnetbuffstatus
```

です。これらの関数は、受信バッファ処理を行う場合に使用できます。
また、次の関数は受信バッファ処理中は使用できません。

```
Lib_cnet_irq_set
Lib_cnet_irq_clear
Lib_cnet_receive_status
Lib_cnet_receive
Lib_gets_cnet
Lib_getc_cnet
```

戻り値

ありません。

例

9.1　ユーザ・プログラム作成例を参照してください。

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnetbuffclose

機　能

受信バッファ処理の終了

形　式

```
#include "f_cnet.h"
void Lib_cnetbuffclose(void);
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信バッファ処理を終了する場合は、必ずこの関数をコールしてください。

戻り値

ありません。

例

9.1　ユーザ・プログラム作成例を参照してください。

留意事項

ありません。

Lib_setcnetrcv

機　能

送信元ノード番号、受信バッファの登録

形　式

```
#include "f_cnet.h"
int Lib_setcnetrcv( int node, char *buff, int size );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

送信元のノード番号と、ノード番号に対応する受信バッファのアドレスと大きさを登録します。
割り込み処理で、登録したノード番号に対応する受信バッファに、受信データを格納します。
登録していないノードからの受信データは捨てます。
送信元が複数（複数台の 90X シリーズが接続されている）の場合は、この関数を必要な回数コールして登録を行います。
この関数は、受信バッファ処理の開始(Lib_cnetbuffopen)後に必ずコールしてください。

① ***node*** は送信先のノード番号です。

値	定　数	意　味
0	NODE_0	送信先ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	送信先ノード番号 63 を指定します。

② ****buff*** は受信バッファのポインタです。

③ ***size*** は受信バッファの大きさです。
64 バイト～16384 バイト

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	引数の値が異常です。

例

9.1　ユーザ・プログラム作成例を参照してください。

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_clrcnetrcv

機　能

送信元ノード番号、受信バッファの登録解除

形　式

```
#include "f_cnet.h"
int Lib_clrcnetrcv( int node );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

送信元のノード番号と受信バッファの登録を解除します。

① ***node*** は送信先のノード番号です。

値	定　数	意　味
0	NODE_0	送信先ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	送信先ノード番号 63 を指定します。
0xFF	NODE_ALL	すべての登録を解除します。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	引数の値が異常です。

例

ノード番号０の登録を解除します。

```
#include "f_cnet.h"

void main( void )
{
    char buff[ 256 ];

    if ( NO_ERROR == Lib_open_cnet( NODE_8) )
    {
        Lib_cnetbuffopen();                     /* 受信バッファ処理の開始 */
        Lib_setcnetrcv( NODE_0, buff, 256 );    /* 受信バッファの登録 */

        /* 何らかの処理が入ります */

        Lib_clrcnetrcv( NODE_0 );               /* ノード番号０の登録解除 */

        /* 何らかの処理が入ります */

        Lib_cnetbuffclose();                    /* 受信バッファ処理の終了 */
        Lib_close_cnet();
    }
}
```

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_getcnetrcv

機　能

登録状況の参照

形　式

```
#include "f_cnet.h"
void Lib_getcnetrcv( unsigned char *map );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

現在登録されているノード番号を参照します。
登録とは、Lib_setcnetrcv 関数による送信元ノード番号、受信バッファの登録を指します。

① ***map*** は登録ノード格納用配列のポインタです。
配列は、unsigned char 型で 64 要素確保してください。
[例] unsigned char map[64];
ノード番号をインデックスとして、配列内容が 0ｘFF の場合は該当ノードが登録済みで、0x00 の場合は登録されていません。

戻り値

ありません。

例

登録状況を参照し、登録済のノード番号を表示します。

```
#include "f_graph.h"
#include "f_cnet.h"

void main( void )
{
    char buff[ 256 ];
    unsigned char map[ 64 ];
    short i;

    if ( NO_ERROR == Lib_open_cnet( NODE_8 ) )
    {
        Lib_cnetbuffopen();                     /* 受信バッファ処理の開始 */
        Lib_setcnetrcv( NODE_0, buff, 256 );    /* 受信バッファの登録 */

        /* 何らかの処理が入ります */

        Lib_getcnetrcv( map );                  /* 登録状況参照 */
        for ( i = 0; i < 64; i++ )
        {
            if ( map[ i ] == 0xff )
                Lib_chrdisp((1+i),10,"x");
        }

        Lib_cnetbuffclose();                    /* 受信バッファ処理の終了 */
        Lib_close_cnet();
    }
}
```

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnetbuffcheck

機　能

受信バッファ内有効データ長の取得

形　式

```
#include "f_cnet.h"
int Lib_cnetbuffcheck( int node );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信バッファ内の有効データ長を取得します。

① ***node*** は受信先のノード番号です。

値	定　数	意　味
0	NODE_0	受信先ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先ノード番号 63 を指定します。

戻り値

値	定　数	意　味
-1 以外		有効データ長（バイト）を返します。
-1	ARG_ERR	引数の値が異常です。

例

9.1　ユーザ・プログラム作成例を参照してください。

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnetbuffrcv

機　能

受信バッファから 1 バイトデータの読み込み

形　式

```
#include "f_cnet.h"
int Lib_cnetbuffrcv( int node );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信バッファから読み込んだ 1 バイトのデータを符号拡張せずに、short に変換してから返します。何らかのエラーが発生している場合は-1 を返します。
エラーの内容は Lib_cnetbuffstatus 関数で参照できます。

① ***node*** は受信先のノード番号です。

値	定　数	意　味
0	NODE_0	受信先ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先ノード番号 63 を指定します。

戻り値

値	定　数	意　味
-1 以外		受信データを返します。
-1		何らかのエラーが発生しています。

例

9.1　ユーザ・プログラム作成例を参照してください。

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnetbuffclr

機　能

受信バッファの初期化

形　式

```
#include "f_cnet.h"
int Lib_cnetbuffclr( int node );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

受信バッファを初期化します。すでに受信しているデータは無くなります。

① ***node*** は受信先のノード番号です。

値	定　数	意　味
0	NODE_0	受信先ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	受信先ノード番号 63 を指定します。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	引数の値が異常です。

例

エラーが発生している場合、エラーコードを表示し受信バッファを初期化します。

```
#include "f_stdio.h"
#include "f_cnet.h"

void main( void )
{
    char buff[ 256 ];
    short ret;

    if ( NO_ERROR == Lib_open_cnet( NODE_8) )
    {
        Lib_cnetbuffopen();                     /* 受信バッファ処理の開始 */
        Lib_setcnetrcv( NODE_0, buff, 256 );    /* 受信バッファの登録 */

        /* 何らかの処理が入ります */

        ret = Lib_cnetbuffstatus( NODE_0 );     /* エラーステータスの取得 */
        if ( NO_ERROR != ret )
        {
            Lib_printf( "%d ¥n", ret );
            Lib_cnetbuffclr( NODE_0 );          /* 受信バッファのクリア */
        }

        Lib_cnetbuffclose();                    /* 受信バッファ処理の終了 */
        Lib_close_cnet();
    }
}
```

留意事項

○ Lib_open_cnet を実行前にコールする必要があります。

Lib_cnetbuffstatus

機　能

エラーステータスの取得

形　式

```
#include "f_cnet.h"
int Lib_cnetbuffstatus( int  node );
```

901	902	903	904	FVL/LNX	高分解能
○	○				○

解　説

エラーが発生しているか調べます。
エラーが発生すると、ノードごとにエラー標識がセットされます。
エラー標識はこの関数を呼び出すまでセットされたままで、呼び出すことによりクリアされます。

① ***node*** は送受信先のノード番号です。

値	定　数	意　味
0	NODE_0	送受信先ノード番号 0 を指定します。
▪	▪	▪
63	NODE_63	送受信先ノード番号 63 を指定します。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	引数の値が異常です。
-2	NO_CONNECT	まだ接続されていません。
888	HW_ERROR	インターフェイスまたは制御関数に異常が発生しました。
999	UN_KNOWN	インターフェイスまたは制御関数に未知の異常が発生しました。
20	NOT_REGIST	ノードが登録されていません。
21	BUFF_OVER	受信バッファオーバフローしました。
22	BUFF_DATA_NOTHING	受信バッファにデータがありません。

例

Lib_cnetbuffclr を参照してください。

留意事項

ありません。

9.1　ユーザ・プログラム作成例

（受信バッファ処理を使用した、割り込みによる受信）

2 台の 901（ノード 9, 10）からのデータを受信し、受信データを表示します。

例

```
#include <string.h>
#include "f_stdio.h"
#include "f_graph.h"
#include "f_cnet.h"

#define HOST901_NODE    8
#define NO1_901_NODE    9
#define NO2_901_NODE   10

static unsigned char message[128];      /* 送信用 */
static unsigned char data[128];         /* 送信用 */

static void cnet_err_msg( int );


void main( void )
{
    int ret;                    /* 関数戻り値 */
    unsigned char c;
    int len, cmdlen;
    int i;
    char tc;                     /* 送信文字 */
    int  sw;                     /* 送信先切り替え用スイッチ 0:No1 1:No2 */
    char buff[10];
    static char no1buff[ 256 ];
    static char no2buff[ 256 ];
    static char string[10];
    static char prestr[10];
    unsigned char map[ 64 ];
    int rcvcnt;
    int rcvdata;

    tc = 'A';                    /* 送信文字初期値 */
    sw = 0;                      /* スイッチ初期値(No1 901) */

    Lib_openkey();
    Lib_keyinput( WAITING );     /* キー入力で開始 */
    Lib_time_delay( 500 );
    /* C-NET の初期化 */
    ret = Lib_open_cnet( HOST901_NODE );
    if ( !ret )                  /* 正常終了 ? */
    {                           /* Yes */
        /* 受信バッファ処理の開始 */
        Lib_cnetbuffopen();

        /* 901 送信元ノード番号､受信バッファの登録 */
        ret = Lib_setcnetrcv( NO1_901_NODE, no1buff, 256 );
```

```
if ( ret != 0 )
    Lib_invdisp( 1, 1, "No1 setcnetrcv error" );

/* 901 送信元ノード番号､受信バッファの登録 */
ret = Lib_setcnetrcv( NO2_901_NODE, no2buff, 256 );
if ( ret != 0 )
    Lib_invdisp( 1, 3, "No2 setcnetrcv error" );

/* 登録状況の参照 */
Lib_getcnetrcv( map );
for ( i = 0; i < 25; i++ )
{
    if( 0xff == map[i] )
    {
        Lib_sprintf( string, "map node[%d]", i );
        Lib_chrdisp( 1, (8+i), string );
    }
}
memset(message, 0x1a, 128);        /* C-NET 送信用バッファの初期化 */
memset(data, 0x1a, 2048);    /* C-NET 送信用バッファの初期化 */

for ( ; ; )
{
    data[ 0 ] = tc;          /* 送信文字をデータパケットに格納 */
    tc += 1;                 /* 送信文字を変更する */
    if ( tc == 'Z' + 1 )     /*    (A～Z)       */
        tc = 'A';            /*                    */
    if ( !sw )               /* No1 901 に送信 ? */
    {
        sw = 1;              /* Yes, 次回 No2 に送信 */
                             /* No1 901 に 1 文字送信 */
        ret = Lib_puts_cnet( data, 1, TEXT_TYPE, NO1_901_NODE );
        if ( ret != NO_ERROR )/* 異常終了 */
        {
            /* No */
            cnet_err_msg( ret );/* エラーメッセージを表示し終了 */
            break;
        }
    }
    else                          /* No2 901 に送信 */
    {
        sw = 0;                       /* 次回 No1 に送信 */
                                      /* No2 901 に 1 文字送信 */
        ret = Lib_puts_cnet( (UBYTE *)&tc, 1, TEXT_TYPE, NO2_901_NODE );
        if ( ret != NO_ERROR )        /* 正常終了 */
        {
            /* No */
            cnet_err_msg( ret );
            /* エラーメッセージを表示し終了 */
            break;
        }
        /* 受信バッファ内有効データ長の取得 */
        rcvcnt = Lib_cnetbuffcheck( NO1_901_NODE );

        if ( rcvcnt == 13 )
       {
```

```
                    Lib_chrdisp( 1, 25, "No1 901 : " );
                    for ( i = 0; i < 13; i++)
                    {
                        /* 受信バッファから1バイトデータの読み込み */
                        rcvdata = Lib_cnetbuffrcv( NO1_901_NODE );
                        if ( rcvdata == -1 )
                        {
                            Lib_invdisp( 1, 26, "cnetbuffrcv error" );
                            /* エラーの参照とエラー標識のリセット */
                            ret = Lib_cnetbuffstatus( NO1_901_NODE );
                        }
                        Lib_sprintf( string, "%c", (char)rcvdata );
                        Lib_invdisp( 11 + i ,25,  string );
                        Lib_time_delay( 5 );
                        Lib_chrdisp( 11 + i ,25,  string );
                    }
                }
                /* 受信バッファ内有効データ長の取得 */
                rcvcnt = Lib_cnetbuffcheck( NO2_901_NODE );
                if ( rcvcnt == 13 )
                {
                    Lib_chrdisp( 1, 28, "No2 901 : " );
                    for ( i = 0; i < 13; i++)
                    {
                        /* 受信バッファから1バイトデータの読み込み */
                        rcvdata = Lib_cnetbuffrcv( NO2_901_NODE );
                        if ( rcvdata == -1 )
                        {
                            Lib_invdisp( 1, 29, "cnetbuffrcv error" );
                            ret = Lib_cnetbuffstatus( NO2_901_NODE );
                        }
                        Lib_sprintf( string, "%c", (char)rcvdata );
                        Lib_invdisp( 11 + i ,28,  string );
                        Lib_time_delay( 5 );
                        Lib_chrdisp( 11 + i ,28,  string );
                    }
                }
            if ( NOT_KEYIN != Lib_keyinput( NO_WAIT) )   /* キー入力で終了 */
                break;
        }
        Lib_cnetbuffclr( NODE_ALL );
        Lib_clrcnetrcv( NODE_ALL );
        Lib_cnetbuffclose();
        Lib_close_cnet();                /* C-NETの終了 */
        Lib_closekey();
    }
}

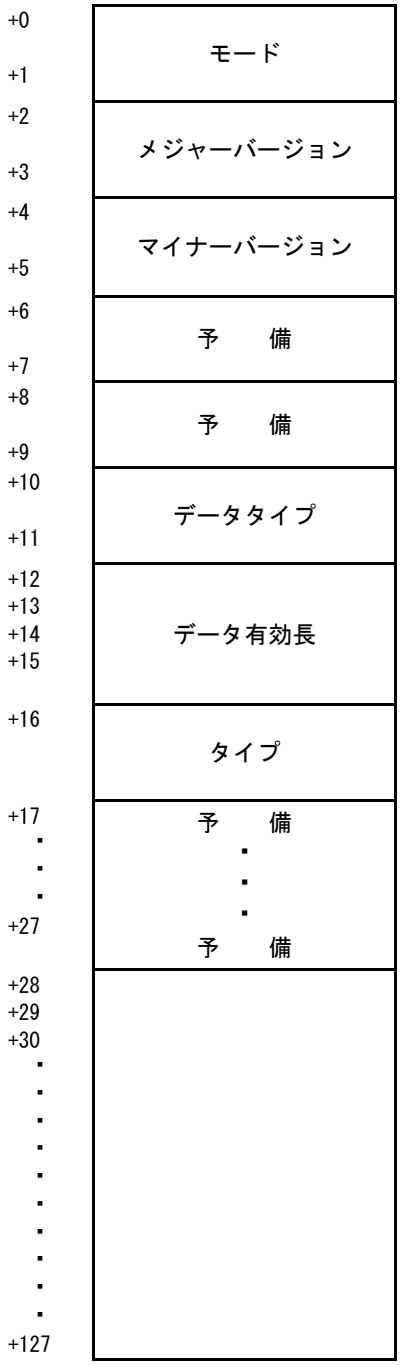

/* C-NETのエラーメッセージを表示する。*/
static void cnet_err_msg( int ret )
{
    switch(ret)
    {
            case NOT_READY:
                Lib_invdisp( 1, 30,"C-NET Error : NOT_READY");
```

```
            break;
        case ARG_ERR:
            Lib_invdisp( 1, 30,"C-NET Error : ARG_ERR");
            break;
        case HW_ERROR:
            Lib_invdisp( 1, 30,"C-NET Error : HW_ERROR");
            break;
        case UN_KNOWN:
            Lib_invdisp( 1, 30,"C-NET Error : UN_KNOWN");
            break;
        case ALREADY_EXIST:
            Lib_invdisp( 1, 30,"C-NET Error : ALREADY_EXIST");
            break;
        case LINE_BUSY:
            Lib_invdisp( 1, 30,"C-NET Error : LINE_BUSY");
            break;
        case NO_CONNECT:
            Lib_invdisp( 1, 30,"C-NET Error : NO_CONNECT");
            break;
        case SEND_TO_ME:
            Lib_invdisp( 1, 30,"C-NET Error : SEND_TO_ME");
            break;
        case NOT_COMPLETE:
            Lib_invdisp( 1, 30,"C-NET Error : NOT_COMPLETE");
            break;
        case NO_ACK:
            Lib_invdisp( 1, 30,"C-NET Error : NO_ACK");
            break;
        default:
            Lib_invdisp( 1, 30,"??????????");
    }
}
```

9.2　データパケット構造

C-NET で通信を行うためのデータパケット構造について説明します。
データパケットは「メッセージ･フィールド」と「データ･フィールド」で構成されます。

9.3　メッセージフィールド構造



送信モードを指定します。
[0x0000] 901 システムの転送
[0x0001] 上記以外

901 システム転送時、システムのバージョンを ASCII で格納
［例］1.23

'1'	' '	'2'	'3'
+2	+3	+4	+5

空きは NULL で埋めます。

901 システム転送以外は 0x1a で埋めます。

[0x000] 下記以外
[0x001] システム、ユーザファイル転送時

データフィールドの有効長

データフィールドの内容
[0x00]ASCII[0x01] バイナリー

ファイル転送時、データフィールドの種別を指定します。

'S'	'T'	'A'	'¥0'	データパケットがヘッダの時（ファイル転送の開始）
+28	+29	+30	+31	
'D'	'A'	'T'	'¥0'	データパケットがファイル本体時
+28	+29	+30	+31	
'E'	'N'	'D'	'¥0'	ファイル転送の終了
+28	+29	+30	+31	

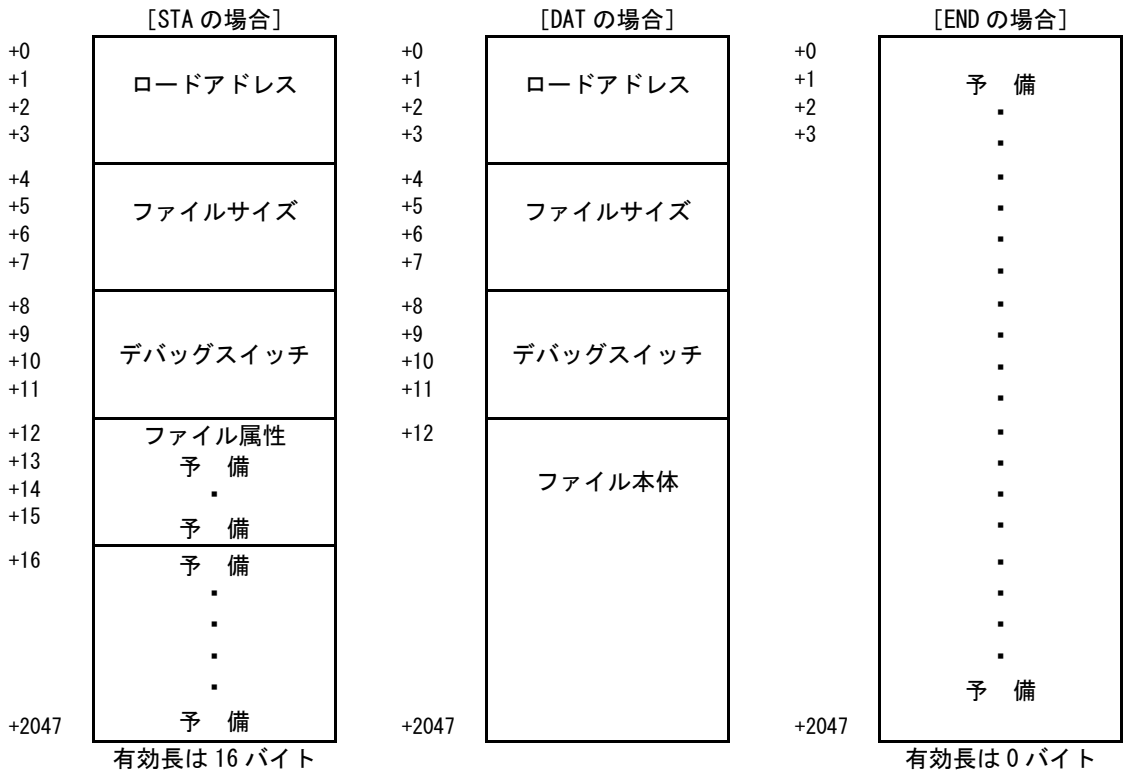
ファイル転送以外は 0x1a で埋めます。

予備、空きデータは 0x1a で埋めます。

9.4　データフィールド構造

(1) システムファイル転送

予備、空きデータは 0x1a で埋めます。



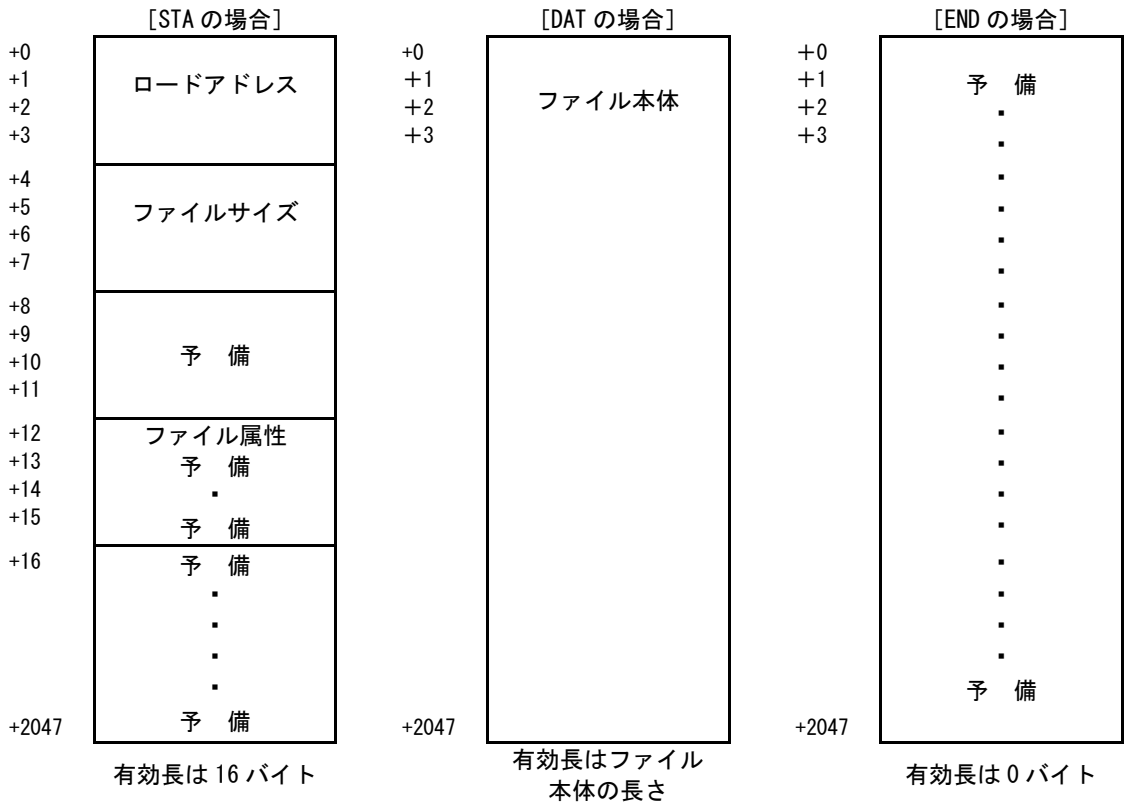
○ 901 システムファイルの先頭 16 バイトにはロードアドレス、ファイルサイズ、デバッグスイッチ、ファイル属性が含まれていて以後、システム本体となっています。
STA の先頭 16 バイトには、901 システムファイルの先頭 16 バイトと同じものを格納します。

○ DAT のロードアドレス、ロードサイズ、デバッグスイッチは先頭の DAT にのみ含めます。
これらは、901 システムファイルの先頭 16 バイトからファイル属性を除いたものです。
以後はファイル本体の続きになります。

○ ホスト 901 間のファイル転送は「ファイルシステムライブラリ」の Lib_fcopy を使用してください。

(2) ユーザーファイル転送

予備、空きデータは 0x1a で埋めます。

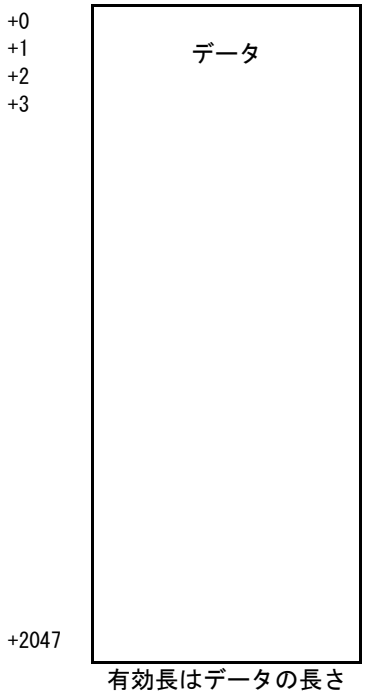


○ 90X シリーズ用のユーザファイル先頭 16 バイトには、ロードアドレス、ファイルサイズ、予備、ファイル属性が含まれ、以後ファイル本体となります。
STA の先頭 16 バイトには、90X シリーズ用のユーザーファイル先頭 16 バイトと同じものを格納します。

○ ホスト 90X シリーズ間のファイル転送は「ファイルシステムライブラリ」の
Lib_fcopy を使用してください。

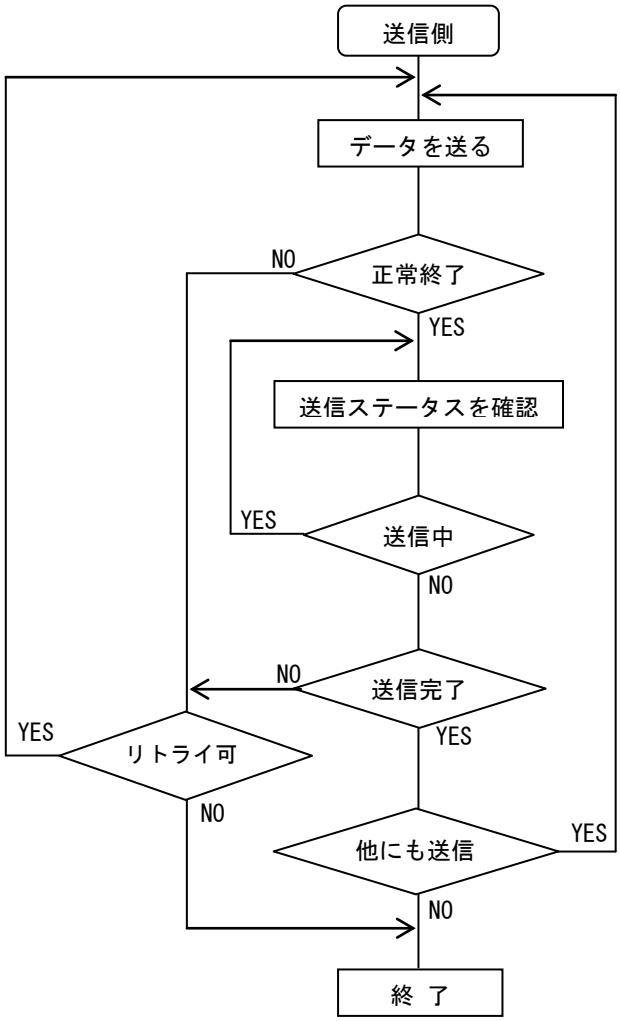
(3) ファイル転送以外

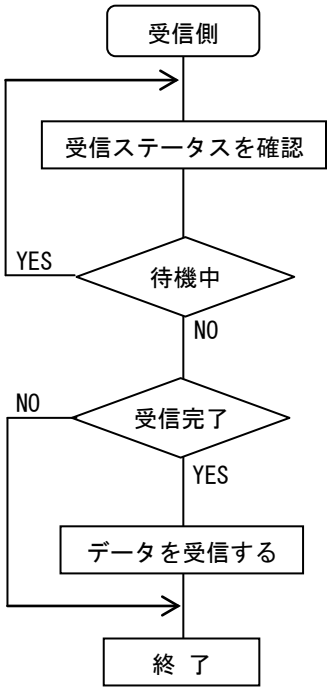
予備、空きデータは、0x1a で埋めます。



9.5　通信手順

ファイル転送、データの転送共、次の方法で通信できます。
ファイル転送時は最初に STA のパケットを送り、次に DAT のパケットがなくなるまで送ります。
最後に END のパケットを送りファイル転送の終了となります。





10. C-LINK ライブラリ

C-LINK は、90X シリーズとホスト間をつなぐホスト通信インターフェイスで、下記の特徴があります。
なお、本ライブラリは、901,901a ではサポートしておりません。

① パソコンに標準装備されている RS232C シリアルポートを利用して通信します。
C-NET のような特殊なインターフェイスボードを必要としません。
従ってノートパソコンのように拡張ボードが実装できないパソコンでも C-LINK は使えます。

② ファースト標準のシリアルケーブルがそのまま利用できます。

③ 通信速度は C-NET より劣りますが、通常の通信速度(9600bps 程度)より大幅に向上しています。
具体的な通信速度は、パソコンの性能にも依存しますが、50Kbps～100Kbps です。

Lib_open_clink

機　能

C-LINK のオープン

形　式

```
#include "f_cnet.h"
int  Lib_open_clink( int port_no, int node_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○		○

解　説

C-LINK のポート番号、ノード番号を指定し、初期化を行います。

① ***port_no*** はポート番号(RS232C チャネル番号)です。(0,1)

② ***node_no*** は自局ノード番号(8～63)です。ノード番号は C-NET と共通です。

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	パラメータの値が範囲内にありません。

例

```
#include "f_cnet.h"
#include "f_pinf.h"

unsigned char data[2048];
unsigned char mess[128];

void main( void )
{
    int ret;
    int node;
    int len;
    int f_type;

    /* C-LINK の初期化 */
    ret = Lib_open_clink( Lib_get_clink_port_no(), Lib_get_cnet_node_no() );
    if ( !ret )                /* 正常終了 ? */
    {                                  /* Yes */
        memset(mess, 0, 128);       /* C-LINK 送信用バッファの初期化 */
        memset(data, 0x1a, 2048);   /* C-LINK 送信用バッファの初期化 */

        for ( ; ; )
        {
            /*受信*/
            ret = Lib_gets_clink( data, &len, &f_type, &node,
                                              Lib_get_clink_port_no() );
            if ( ret == NO_ERROR )    /* 正常終了 */
            {                                          /* No */
                /*受信した文字列を返信*/
                ret = Lib_puts_clink( data, len, f_type, node,
                                                Lib_get_clink_port_no() );
                if ( ret != NO_ERROR )        /* 正常終了 */
                    break;
            }
            else
                break;
        }
        Lib_close_clink( Lib_get_clink_port_no() );   /* C-LINK の終了 */
    }
}
```

留意事項

○ メッセージ、データフィールド構造は C-NET と共通です。
○ チャネル番号 0 は C-LINK、チャネル番号 1 は通常のシリアルという使い方もできます。

Lib_close_clink

機　能

C-LINK のクローズ

形　式

```
#include "f_cnet.h"
int Lib_close_clink( int port_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○		○

① ***port_no*** はポート番号(RS232C チャネル番号)です。(0,1)

解　説

C-LINK 接続を解除します。
C-LINK 接続を解除した場合、再び初期化されるまで、C-LINK は利用できません。

戻り値

処理結果

値	定　　数	意　　味
0	NO_ERROR	正常終了しました。
-2	NO_CONNECT	初期化が行われていません。 (接続していません。)
-1	ARG_ERR	パラメータの値が範囲内にありません。

例

Lib_open_clink 関数の例を参照してください。

留意事項

○ メッセージ、データフィールド構造は C-NET と共通です。

Lib_gets_clink

機　能

C-LINK データ受信

形　式

```
#include "f_cnet.h"
int Lib_gets_clink( unsigned char *data, int *length,
                                int *f_type, int *node_no, int port_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○		○

解　説

指定ポート番号からデータを受信します。
受信データ格納用バッファの容量は 2048 バイト分必要です。

① ***data*** は受信データ格納用バッファへのポインタです。

② ***length*** は受信データ長(1～2048)を返します。

③ ***f_type*** は受信ファイルタイプを返します。

値	定　数	意　味
0	TEXT_TYPE	テキスト･タイプのファイルを受信しました。
1	BINARY_TYPE	バイナリ･タイプのファイルを受信しました。

④ ***node_no*** はデータを送信したノード番号です。

⑤ ***port_no*** はポート番号(RS232C チャネル番号)です。(0,1)

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-2	NO_CONNECT	初期化が行われていません。 (接続していません。)
4	RX_NOTHING	受信を完了していません。
9999	IO_TIMEOUT	I/O タイムアウト時間エラーが発生しました。

例

Lib_open_clink 関数の例を参照してください。

留意事項

○ メッセージ、データフィールド構造は C-NET と共通です。

Lib_getc_clink

機　能

C-LINK1 文字受信

形　式

```
#include "f_cnet.h"
int Lib_getc_clink( int *node_no, int port_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○		○

解　説

1 バイトを指定のポート番号から入力します。

① ***node_no*** はデータを送信したノード番号です。

② ***port_no*** はポート番号(RS232C チャネル番号)です。(0,1)

戻り値

入力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

```
#include "f_cnet.h"

void main( void )
{
    int ret;
    char ch;
    int node;

    /* C-LINK の初期化 */
    ret = Lib_open_clink( Lib_get_clink_port_no(), Lib_get_cnet_node_no() );
    if ( !ret )                /* 正常終了 ? */
    {                                  /* Yes */
        for ( ; ; )
        {
            /*受信*/
            ret = Lib_getc_clink( &node, Lib_get_clink_port_no() );
            if ( ret != EOF ) /* 正常終了 */
            {
                ch = (char)(ch & 0x000000ff);
                /*受信した文字列を返信*/
                ret = Lib_putc_clink( ch, node, Lib_get_clink_port_no() );
                if ( ret == EOF )      /* 正常終了 ?*/
                    break;
            }
            else
                break;
        }
        Lib_close_clink( Lib_get_clink_port_no() );  /* C-LINK の終了 */
    }
}
```

留意事項

○ メッセージ、データフィールド構造は C-NET と共通です。

Lib_puts_clink

機　能

C-LINK データ送信

形　式

```
#include "f_cnet.h"
int Lib_puts_clink( unsigned char *data, int length,
                                int f_type, int node_no, int port_no  );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○		○

解　説

C-LINK が指定されたノード番号にデータを送信します。
メッセージ・フィールドに規定値をセット後送信しています。

① ***data*** は送信データバッファへのポインタです。

② ***length*** は送信データ長（1～2048）です。

③ ***f_type*** は送信ファイルタイプです。

値	定　数	意　味
0	TEXT_TYPE	テキスト･タイプのファイルを送信します。
1	BINARY_TYPE	バイナリ･タイプのファイルを送信します。

④ node_no は送信先のノード番号です。

⑤ port_no はポート番号(RS232C チャネル番号)です。(0,1)

戻り値

値	定　数	意　味
0	NO_ERROR	正常終了しました。
-1	ARG_ERR	パラメータの値が範囲内にありません。
-2	NO_CONNECT	初期化が行われていません。 (接続していません。)
-5	SEND_TO_ME	ノード番号が自局の番号と同一です。
4	RX_NOTHING	受信を完了していません。
5	NOT_COMPLETE	先の送信が完了していません。
6	LINE_BUSY	送信ポートの状態がビジィーです。
9999	IO_TIMEOUT	I/O タイムアウト時間エラーが発生しました。

例

Lib_open_clink 関数の例を参照してください。

留意事項

○ メッセージ、データフィールド構造は C-NET と共通です。

Lib_putc_clink

機　能

C-LINK1 文字送信

形　式

```
#include "f_cnet.h"
int  Lib_putc_clink( char ch, int node_no, int port_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○		○

解　説

指定したポート番号から 1 バイトを送信します。

① ***ch*** は出力文字です。

② ***node_no*** は送信先のノード番号です。

③ ***port_no*** はポート番号(RS232C チャネル番号)です。(0,1)

戻り値

出力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

Lib_getc_clink 関数の例を参照してください。

留意事項

○ メッセージ、データフィールド構造は C-NET と共通です。

11. パラレル・リンクライブラリ

パラレル・リンクは、902 シリーズとホスト間をつなぐホスト通信インターフェイスで、下記の特徴があります。なお、本ライブラリは、901,903,904 シリーズではサポートしておりません。

① パソコンに標準装備されているパラレル(プリンター)ポートを利用して通信します。
C-NET のような特殊なインターフェイスボードを必要としません。

② 専用のケーブルが必要です。

③ 通信速度は、パソコンの性能にも依存しますが C-LINK より高速です。

Lib_open_plink

機　能

パラレル･リンクの初期化

形　式

```
#include "f_plink.h"
int Lib_open_plink( void );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

パラレル･リンクの初期化を行います。
パラレル･リンクを使用する際には、必ず実行してください。

戻り値

処理結果

値	定　数	意　味
0	NO_ERROR	常に正常終了。

例

文字列を受信し、受信した文字列をそのまま送信します。

```
#include "f_plink.h"

unsigned char ucData[ 2048 ];


void main( void )
{
    int  iLength;

    if ( NO_ERROR == Lib_open_plink() )
    {
        while( 1 )
        {
            if ( COMPLETE == Lib_receive_status_plink() )
            {  /* 受信完了した場合 */
                if ( NO_ERROR == Lib_gets_plink( ucData, &iLength ) )
                {  /* 受信に成功した場合 */
                    if ( NO_ERROR == Lib_puts_plink( ucData, iLength ) )
                    {  /* 受信した文字列を送信 */
                        break;
                    }
                }
            }
        }

        Lib_close_plink()
    }
}
```

留意事項

ありません。

Lib_close_plink

機　能

パラレル･リンクの終了

形　式

```
#include "f_plink.h"
int Lib_close_plink( void );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

パラレル･リンクを終了します。

戻り値

処理結果

値	定　　数	意　　味
0	NO_ERROR	正常終了。
-2	NO_CONNECT	パラレル･リンクが初期化されていない。

例

Lib_open_plink 関数の例を参照してください。

留意事項

ありません。

Lib_receive_status_plink

機　能

受信ステータスの取得

形　式

```
#include "f_plink.h"
int Lib_receive_status_plink( void );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

受信の状態を調べて報告します。

戻り値

処理結果

値	定　　数	意　　味
0	COMPLETE	受信完了しました。
3	READY	受信していません。
-2	NO_CONNECT	初期化されていません。
888	HW_ERROR	インターフェースまたは制御関数に異常が発生しました。

例

Lib_open_plink 関数の例を参照してください。

留意事項

ありません。

Lib_gets_plink

機　能

文字列を受信

形　式

```
#include "f_plink.h"
int Lib_gets_plink( unsigned char *data, int *length );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

文字列を受信します。受信データには NULL は付加されません。

① ***data*** は受信データを格納するバッファのポインタです。
バッファは 2048 バイト必要です。

② ***length*** は受信データ長を格納する変数のポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NO_ERROR	正常終了しました。
-2	NO_CONNECT	初期化されていません。
999	IO_TIMEOUT	I/O タイムアウト時間が発生しました。

例

Lib_open_plink 関数の例を参照してください。

留意事項

ありません。

Lib_getc_plink

機　能

1 文字受信

形　式

```
#include "f_plink.h"
int Lib_getc_plink( void );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

1 文字(1 バイト)受信します。

戻り値

入力した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

文字を受信し、受信した文字をそのまま送信します。

```
#include "f_plink.h"

void main( void )
{
    int  iData;

    if ( NO_ERROR == Lib_open_plink() )
    {
        while( 1 )
        {
            if ( COMPLETE == Lib_receive_status_plink() )
            {  /* 受信完了した場合 */
                iData = Lib_getc_plink();
                if ( EOF != iData )
                {  /* 受信に成功した場合 */
                    iData = Lib_putc_plink( ( char )( iData & 0xff ) );
                    if ( EOF != iData )
                    {
                        break;
                    }
                }
            }
        }

        Lib_close_plink()
    }
}
```

留意事項

ありません。

Lib_puts_plink

機　能

文字列を送信

形　式

```
#include "f_plink.h"
int Lib_puts_plink( unsigned char *data, int length );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

文字列を送信します。

① ***data*** は送信データを格納してあるバッファのポインタです。

② ***length*** は送信データ長です(0～2048)。

戻り値

処理結果

値	定　　数	意　　味
0	NO__ERROR	正常終了しました。
-1	ARG__ERR	引数が異常です。
-2	NO__CONNECT	初期化されていません。
999	IO__TIMEOUT	I/O タイムアウト時間時間が発生しました。

例

Lib_open_plink 関数の例を参照してください。

留意事項

ありません。

Lib_putc_plink

機　能

1 文字送信

形　式

```
#include "f_plink.h"
int Lib_putc_plink( char ch );
```

901	902	903	904	FVL/LNX	高分解能
	○				○

解　説

1 文字(1 バイト)送信します。

① ***ch*** は出力文字です。

戻り値

送信した文字を int の値として返します。
エラーが発生すると EOF を返します。

例

Lib_getc_plink 関数の例を参照してください。

留意事項

ありません。

12. E-LINK ライブラリ

E-LINKは、FV904及びLNXシリーズとホスト間をつなぐホスト通信インターフェイスで、
下記の特徴があります。
なお、本ライブラリは、FV904及びLNXシリーズのみのサポートです。

FV904 において本ライブラリを使用するにはオプションの E-LINK ユニット(※)が必要です。

① Ethernet を使用し、プロトコルには TCP/IP を使用します。

② 通信速度はネットワークの構成にもよりますが、下記の様になります。

FV904：最大230Kbps(理論値)

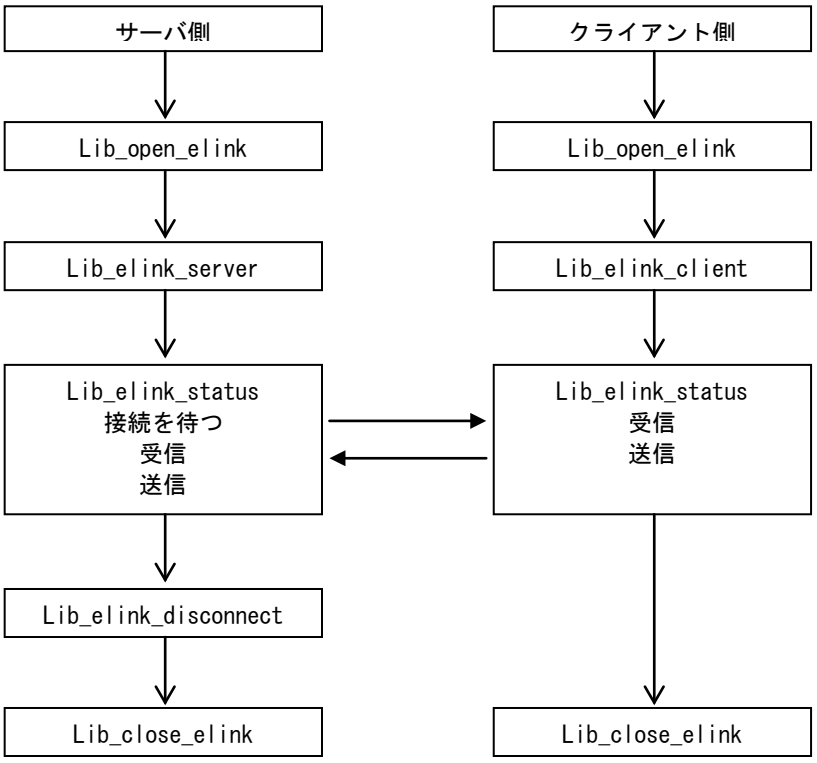
LNXシリーズ：最大100Mbps(理論値)

③ サーバ、クライアント間の 1 対 1 の通信をサポートします。

(※)FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、2008.04.01 現在、
生産・販売ともに終了しております。

通信手順

E-LINK にはサーバ、クライアントが存在します。
サーバは、クライアントの接続を待ちます。
接続後は、サーバ、クライラントを問わず、データの送受信が行えます。



Lib_elink_disconnect はどちらが行ってもかまいません。

Lib_open_elink

機　能

E-LINK オープン

形　式

```
#include "f_elink.h"
Lib_open_elink( void );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

E-LINK をオープンします。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-1	ERROR_RETURN	異常終了
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例（E-LINK サーバサンプル、E-LINK クライアントサンプル）を参照してください。

留意事項

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_close_elink

機　能

E-LINK クローズ

形　式

```
#include "f_elink.h"
Lib_close_elink( void );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

E-LINK をクローズします。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-1	ERROR_RETURN	異常終了
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例（E-LINK サーバサンプル、E-LINK クライアントサンプル）を参照してください。

留意事項

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_server

機　能

E-LINK サーバ開始

形　式

```
#include "f_elink.h"
int Lib_elink_server( void );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

E-LINK のサーバ機能を開始します。
開始後はサーバとしてクライアントの接続を待機します。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-3	ALREADY_CONNECT	すでに接続されているか、サーバが開始されています。
-4	SERVER_ERROR	サーバーが開始できません
-10	ELINK_TIMEOUT	I/O タイムアウト時間
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例（E-LINK サーバサンプル）を参照してください。

留意事項

共通

○ 標準入出力が E-LINK の場合は、ALREADY_CONNECT が返却されます。

○ ALREADY_CONNECT の場合、Lib_elink_status 関数で接続しているか確認してください。

○ SERVER_ERROR, ELINK_TIMEOUT が発生した場合は、電源を再投入してください。

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

○ 通信切断後、すぐに、Lib_elink_server 関数をコールする場合は、2 秒程度かかる場があります。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_client

機　能

E-LINK クライアント開始

形　式

```
#include "f_elink.h"
int Lib_elink_client( unsigned int Ip, unsigned short Port);
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

E-LINK のクライアント接続を行います。

① ***Ip*** は接続先の IP アドレスです。
上位から 8 ビットずつの入力となります。
例. 192.168.200.254 の場合、 0xC0A8C8FE となります。
0.0.0.0、255.255.255.255 は指定できません。

② ***Port*** は接続先のポート番号です。(0～65535)

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了。
-1	ERROR_RETURN	パラメータエラー
-3	ALREADY_CONNECT	すでに接続されているか、 既にサーバーとして起動している
-5	CLIENT_ERROR	接続できなかった
-10	ELINK_TIMEOUT	I/O タイムアウト時間
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例 (E-LINK クライアントサンプル)を参照してください。

留意事項

共通

○ クライアント接続はサーバから切断後すぐに実行すると、CLIENT_ERROR が発生する場合があります。(ネットワークの仕様のため)
CLIENT_ERROR が発生した場合は、時間を置いて(2 秒程度)関数コールしてください。

○ ELINK_TIMEOUT が発生した場合は、電源を再投入してください。

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

○ E-LINK ユニットは、クロスケーブルで接続時、対向する 2 台の電源投入のタイミングが数百 msec ずれると、稀ではありますが、リンク確立に若干の時間(数秒～1 分程度)が、かかる場合があります。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_status

機　能

E-LINK 状態監視

形　式

```
#include "f_elink.h"
int Lib_elink_status( int *Connect, int *Recv );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

現在の E-LINK の状態を監視します。

① ***Connect*** は接続状態を得ます。
　　0：接続なし
　　1：接続

② ***Recv*** は受信状態を得ます。
　　0：受信なし
　　1：受信データあり
　受信データがある場合は、Lib_elink_receive を使用してデータを受信してください。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-1	ERROR_RETURN	異常終了
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例（E-LINK サーバサンプル､E-LINK クライアントサンプル）を参照してください。

留意事項

共通

○ 受信データがある場合は、Lib_elink_receive で直ちに受信処理を行ってください。
　受信処理を行わないと、次の受信データで、上書きされる場合があります。

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_receive

機　能

E-LINK 受信

形　式

```
#include "f_elink.h"
int Lib_elink_receive( char *OutPtr, int *DataSize );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

接続先からのデータを受信します。

① ***OutPtr*** は受信データが入るバッファの先頭アドレスです。

② ***DataSize*** は受信したデータのサイズです。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-1	ERROR_RETURN	異常終了
-2	NO_CONNECT	接続されていない
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例（E-LINK サーバサンプル、E-LINK クライアントサンプル）を参照してください。

留意事項

共通

○ 受信サイズは最大 1024 バイトです。
　バッファは 1024 バイト以上確保してください。

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_send

機　能

E-LINK 送信

形　式

```
#include "f_elink.h"
int Lib_elink_send( char *InPtr, int DataSize );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

接続先にデータを送信します。

① ***InPtr*** は送信データのバッファの先頭アドレスです。

② ***DataSize*** は送信データのサイズを指定します。(1～1024)

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-1	ERROR_RETURN	バッファのサイズが 1024 バイトを超えている
-2	NO_CONNECT	接続されていない
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例 (E-LINK サーバサンプル､E-LINK クライアントサンプル) を参照してください。

留意事項

共通

○ 受信サイズは最大 1024 バイトです。
　バッファは 1024 バイト以上確保してください。

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_get_client

機　能

E-LINK 情報取得

形　式

```
#include "f_elink.h"
int Lib_elink_get_client( unsigned int *IP, unsigned short *Port );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

E-LINK に接続したクライアント(クライアントの場合、接続先)の情報を得ます。

① ***IP*** は宛先の IP アドレスです。

② ***Port*** は宛先のポート番号です。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-2	NO_CONNECT	接続されていない
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例 (E-LINK サーバサンプル)を参照してください。

留意事項

共通

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

Lib_elink_disconnect

機　能

E-LINK 切断

形　式

```
#include "f_elink.h"
int Lib_elink_disconnect( void );
```

901	902	903	904	FVL/LNX	高分解能
			△	○	

解　説

接続しているクライアントもしくはサーバーを切断します。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了
-2	NO_CONNECT	接続されていない
-10	ELINK_TIMEOUT	I/O タイムアウト時間
-20	ELINK_NOT_OPEN	オープンされていません
-91	NO_DEVICE	E-LINK ユニットがありません

例

12.1　ユーザープログラム作成例（E-LINK サーバサンプル､E-LINK クライアントサンプル）を参照してください。

留意事項

共通

○ サーバとして起動し、接続しないで終了させるときは、この関数をコールしてください。

○ ELINK_TIMEOUT が発生した場合は、電源を再投入してください。

○ ELINK_NOT_OPEN が発生したときは、Lib_elink_open 関数をコールしてください。

○ 通信接続後、すぐにこの関数をコールせずに、1 秒程度間をあけてください。
すぐにコールした場合、Lib_elink_server で SERVER_ERROR が出る場合があります。

904

○ FV904 において本ライブラリを使用するにはオプションの E-LINK ユニットが必要です。

○ FV904 のオプション製品として E-LINK ユニットを販売致しておりましたが、
2008.04.01 現在、生産・販売ともに終了しております。

12.1　ユーザープログラム作成例

```
/*  E-LINK サーバサンプル                              */
/*  クライアントからの送信をループバックします  */

#include "f_stdio.h"
#include "f_graph.h"
#include "f_elink.h"
#include "f_gui.h"
#include "f_stdlib.h"

void    main(void)
{
    char    baMessage[128];
    char    baBuffer[1024];
    int     wRecvSize;
    int     wResult;
    int     Connect,Recv;
    static int      StCon = 0;
    unsigned int    uwIP;
    unsigned short  usPort;

    /*  カーソル制御用  */
    int         x,y;

    Lib_init_cursor();  /*  カーソルを初期化        */

    if( NORMAL_RETURN != (wResult = Lib_open_elink()) )     /*  E-LINK のオープン   */
        return;

    if( ALREADY_CONNECT == Lib_elink_server() )     /*  サーバーは既に接続されている？  */
    {
        if( NORMAL_RETURN != Lib_elink_disconnect() )   return;     /*  一度、切断          */
        if( NORMAL_RETURN != Lib_elink_server() )       return;     /*  サーバとして起動    */
    }

    Lib_chrdisp(1,30,"キャンセルで終了");
    x = Lib_get_fx_size()/2;
    y = Lib_get_fy_size()/2;
    for(;;)
    {
        if( CURSOR_CANCEL == Lib_see_current_position(&x,&y) )
            break;

        Lib_elink_status( &Connect, &Recv );

        if(1 == Connect && 0 == StCon)
        {
            /*  接続されたらクライアントの情報を表示    */
            Lib_elink_get_client( &uwIP, &usPort );
            Lib_sprintf( baMessage,"IP -> %3d.%3d.%3d.%3d, PORT -> %d",
                    uwIP>>24,(uwIP>>16)&0xff,(uwIP>>8)&0xff,uwIP&0xff, usPort );
            Lib_chrdisp(1,26,baMessage);
            Lib_chrdisp(1,28,"接続");
```

```
            StCon = 1;
        }
        else if(1 == StCon && 0 == Connect)
        {
            Lib_chrdisp(1,28,"切断");
            while( NORMAL_RETURN != (wResult = Lib_elink_server()))
                                                    /*  切断されたらサーバを起動    */
                if( ELINK_TIMEOUT == wResult )
                    return;
            StCon = 0;
        }

        if( Recv )      /*  受信    */
        {
            Lib_elink_receive( baBuffer, &wRecvSize );  /*  データを受信                */
            Lib_elink_send( baBuffer, wRecvSize );      /*  受信した場合はループバック  */
        }
    }

    Lib_elink_disconnect();     /*  切断            */
    Lib_close_elink();          /*  E-LINK クローズ  */
    return;
}
```

```
/*  E-LINK クライアントサンプル         */
/*  実行キーで文字を送信します  */

#include <string.h>

#include "f_stdio.h"
#include "f_graph.h"
#include "f_elink.h"
#include "f_gui.h"
#include "f_stdlib.h"


void    main(void)
{
    char        baMessage[256];
    char        baBuffer[1024];
    int         wRecvSize;
    int         wResult;
    int         Connect,Recv;

    /*  接続先情報  */
    unsigned int    uwIP = 0xC0A8C8FE;          /*  IP = 192.168.200.254    */
    unsigned short  usPort = 257;

    /*  カーソル制御用  */
    int         x,y;
    int         CurStatus;

    Lib_init_cursor();  /*  カーソルを初期化        */

    if( NORMAL_RETURN != Lib_open_elink() )
        return;

    wResult = Lib_elink_server();
    /*  既に接続されている場合 もしくは サーバオープン済みの場合    */
    if( ALREADY_CONNECT == wResult)
    {
        Lib_elink_disconnect();             /*  一度、接続を解除        */
        if( NORMAL_RETURN != (wResult = Lib_elink_client( uwIP, usPort )) )
            return;
    }

    Lib_chrdisp(1,30,"キャンセルで終了");
    x = Lib_get_fx_size()/2;
    y = Lib_get_fy_size()/2;
    for(;;)
    {
        CurStatus = Lib_see_current_position(&x,&y);
        if( CURSOR_CANCEL == CurStatus )
            break;
        if( CURSOR_EXECUTE == CurStatus )
            Lib_elink_send( "Test Send", strlen("Test Send") );

        Lib_elink_status( &Connect, &Recv );

        if( Recv )      /*  受信    */
```

```
        {
            Lib_elink_receive( baBuffer, &wRecvSize );
            /*  受信したサイズを画面に表示  */
            if( wRecvSize )
            {
                Lib_sprintf( baMessage,"Recv -> %5d Byte",wRecvSize );
                Lib_chrdisp(1,26,baMessage);
            }
        }

    }
    Lib_elink_disconnect();
    Lib_close_elink();
}
```

13. 外部入出力端子ライブラリ

本ライブラリは 901, 904 シリーズに標準でサービスされている外部コントロール端子
(EXT.CONTROL コネクタ)の入出力に関するものです。
なお、外部コントロール端子の詳細は各「取扱説明書」を参照してください。

Lib_external_input

機　能

外部入力端子(4 点)状態参照

形　式

```
#include "f_exio.h"
int Lib_external_input( void );
```

901	902	903	904	FVL/LNX	高分解能
○			○		

解　説

外部入力端子(4 点)状態を参照します。

戻り値

入力状態を下記のビットパタンで返します。

値	定　数	意　味
0x00000001	EX_IO_NO0	入力端子 No.0 を入力しています。
0x00000002	EX_IO_NO1	入力端子 No.1 を入力しています。
0x00000004	EX_IO_NO2	入力端子 No.2 を入力しています。
0x00000008	EX_IO_NO3	入力端子 No.3 を入力しています。

例

入力により処理を実行し、結果を出力します。

```
#include "f_exio.h"

void test()
{
    if ( OFF != ( Lib_external_input( ) & EX_IO_NO0 ))
    {
        /*処理*/
        Lib_external_output( EX_IO_NO0 );
        Lib_alert_output( ON );
    }
}
```

留意事項

901

○ トリガー取り込みモード(Lib_random_video_channel)を設定した場合は使用できません。

904

○ 904(a)でトリガー取り込みモード(Lib_random_video_channel)を設定した場合は使用できません。

○ 904p で露光期間出力設定(Lib_external_set_exposure_output)が「出力許可」になっている場合、外部入力端子を汎用入出力として使用する事は出来ません。「出力禁止」にしてからご使用下さい。

Lib_external_output

機　能

外部出力端子(4 点)状態設定

形　式

```
#include "f_exio.h"
void Lib_external_output( int value );
```

901	902	903	904	FVL/LNX	高分解能
○			○		

解　説

外部出力端子(4 点)状態を設定する。

① ***value*** は出力値です。

値	定　数	意　味
0x00000001	EX_IO_NO0	出力端子 No.0 を出力します。
0x00000002	EX_IO_NO1	出力端子 No.1 を出力します。
0x00000004	EX_IO_NO2	出力端子 No.2 を出力します。
0x00000008	EX_IO_NO3	出力端子 No.3 を出力します。

戻り値

ありません。

例

入力により処理を実行し、結果を出力します。

```
#include "f_exio.h"

void test()
{
    if ( OFF != ( Lib_external_input( ) & EX_IO_NO0 ))
    {
        /*処理*/
        Lib_external_output( EX_IO_NO0 );
        Lib_alert_output( ON );
    }
}
```

留意事項

901

○ トリガー取り込みモード(Lib_random_video_channel)を設定した場合は使用できません。

904

○ 904(a)でトリガー取り込みモード(Lib_random_video_channel)を設定した場合は使用できません。

○ 904p で露光期間出力設定(Lib_external_set_exposure_output)が「出力許可」になっている場合、外部入力端子を汎用入出力として使用する事は出来ません。「出力禁止」にしてからご使用下さい。

Lib_alert_output

機　能

アラート出力制御

形　式

```
#include "f_exio.h"
void Lib_alert_output( int swt );
```

901	902	903	904	FVL/LNX	高分解能
○			○		

解　説

アラート出力を制御します。

① *swt* はスイッチ状態です。

値	定　数	意　味
0	OFF	アラートを出力しません。
1	ON	アラートを出力します。

戻り値

ありません。

例

入力により処理を実行し、結果を出力します。

```
#include "f_exio.h"

void test()
{
    if ( OFF != ( Lib_external_input( ) & EX_IO_N00 ))
    {
        /*処理*/
        Lib_external_output( EX_IO_N00 );
        Lib_alert_output( ON );
    }
}
```

留意事項

ありません。

Lib_external_trigger

機　能

トリガ割り込み関数設定

形　式

```
#include "f_exio.h"
int Lib_external_trigger( void (*function)() );
```

901	902	903	904	FVL/LNX	高分解能
○		○	○		

解　説

外部コントロール(EXT.CONTROL)端子からトリガ信号入力による関数を設定します。

① ***function*** は関数へのポインタです。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

トリガ割り込み関数を設定後、トリガを待ちます。

```
#include "f_exio.h"

static int Status;

void function()
{
    Status = ON;
}

void main()
{

    Status = OFF;
    Lib_external_trigger( function );
    for( ;; )
    {
        if( OFF != Status ) break;
    }
    /*処理*/
}
```

留意事項

○ 外部コントロール端子の詳細は各「取扱説明書」を参照してください。

Lib_external_set_exposure_output

機　能

外部出力端子　露光期間出力設定

形　式

```
#include "f_exio.h"
int Lib_external_set_exposure_output( int mode, unsigned int *uiDelayTime );
```

901	902	903	904	FVL/LNX	高分解能
			△		

解　説

外部出力端子 露光期間出力関連の設定を行います。

① ***mode*** は露光期間出力設定です。「露光期間出力許可」になっている場合ランダムトリガの露光期間中に外部出力端子から信号が出力されます。

値	定　数	意　味
0	なし	露光期間出力禁止
1	なし	露光期間出力許可　(初期値)

② ****uiDelayTime*** は各チャネルの露光期間出力遅延設定へのポインタです。
呼び出し側で unsigned int 4 要素分確保して下さい。
単位はマイクロセックで 0 ～ 75μs の範囲で指定します。

```
uiDelayTime[0] =  チャネル 0 の遅延時間(μs)
uiDelayTime[1] =  チャネル 1 の遅延時間(μs)
uiDelayTime[2] =  チャネル 2 の遅延時間(μs)
uiDelayTime[3] =  チャネル 3 の遅延時間(μs)
```

※ 分解能は 5μs 単位となります。

(例)　74μs と指定　→　実際は 70μs となります。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

露光期間出力設定で露光期間出力を禁止にしてから汎用入出力を使用します。

```
#include "f_exio.h"

void test()
{
    int iMode;
    unsigned int uiDelayTime[4];

    /* 現在の設定値参照 */
    Lib_external_get_exposure_output( &iMode, uiDelayTime );

    iMode = 0;
    /* 露光期間出力禁止設定 */
    Lib_external_set_exposure_output( iMode, uiDelayTime );

    if ( OFF != ( Lib_external_input( ) & EX_IO_NO0 ))
    {
        /*処理*/
        Lib_external_output( EX_IO_NO0 );
        Lib_alert_output( ON );
    }
}
```

留意事項

904

○ 本ライブラリはFV904pのみで使用可能です。

○ 露光期間出力設定が「出力許可」になっている場合、外部入出力端子を汎用入出力として使用する事は出来ません。

○ 初期値は　「露光期間出力許可」になっています。FV904pで外部入出力端子を汎用入出力として使用する場合は必ず本ライブラリで「露光期間出力禁止」に設定して下さい。

○ 外部コントロール端子の詳細は「FV904p取扱説明書」を参照してください。

Lib_external_get_exposure_output

機　能

外部出力端子　露光期間出力設定値 参照

形　式

```
#include "f_exio.h"
int Lib_external_get_exposure_output( int *mode, unsigned int *uiDelayTime );
```

901	902	903	904	FVL/LNX	高分解能
			△		

解　説

外部出力端子 露光期間出力関連の設定値を参照します。

① ***mode*** は露光期間出力設定値へのポインタです。

値	定　数	意　味
0	なし	露光期間出力禁止
1	なし	露光期間出力許可

② ***uiDelayTime*** は各チャネルの露光期間出力遅延設定値へのポインタです。
呼び出し側で unsigned int 4 要素分確保して下さい。
呼び出し後、現在の設定値が下記の様にセットされます。

uiDelayTime[0] = チャネル 0 の遅延時間(μs)
uiDelayTime[1] = チャネル 1 の遅延時間(μs)
uiDelayTime[2] = チャネル 2 の遅延時間(μs)
uiDelayTime[3] = チャネル 3 の遅延時間(μs)

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

現在の露光期間出力設定を参照し、露光期間出力が許可されていたら、
禁止にしてから汎用入出力を使用します。

```
#include "f_exio.h"

void test()
{
    int iMode;
    unsigned int uiDelayTime[4];

    /* 現在の設定値参照 */
    Lib_external_get_exposure_output( &iMode, uiDelayTime );

    if( 1 == iMode )
    {
        iMode = 0;
        /* 露光期間出力禁止設定 */
        Lib_external_set_exposure_output( iMode, uiDelayTime );

        if ( OFF != ( Lib_external_input( ) & EX_IO_NO0 ))
        {
            /*処理*/
            Lib_external_output( EX_IO_NO0 );
            Lib_alert_output( ON );
        }
    }
}
```

留意事項

904

○ 本ライブラリは FV904p で使用可能です。

14. 時刻サービスライブラリ

本ライブラリでは、9.765ms, 0.813802ms の 2 種類のタイム･ベースを用意しています。
必要に応じて使用してください。

[注]901 の場合、以下の注意が必要です。

○ 微細クロックモード以外は、DMA 起動中は、タイマ割り込みが禁止されていますのでカウントされません。

○ 9.765msec をタイムベースとしたタイマは、DMA の走行時にタイマ割込が禁止されるためカウントされません。(Lib_readclk, Lib_sensetc, Lib_time_delay など)

◎ DMA を走行させるライブラリ
☆ Lib_freeze
☆ Lib_strobo_freeze 等のビデオ・フリーズ関連

Lib_strtclk

機　能

実時間クロック初期化

形　式

```
#include "f_time.h"
void Lib_strtclk( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

9.765ms ごとに更新される実時間クロック(ソフトウェア)を 0 に初期化し、再スタートするものです。
この実時間クロックは、ユーザプログラムで読取ることもできます。

戻り値

ありません。

例

何らかの処理の処理時間を計測します。

```
#include "f_time.h"
#include "f_stdio.h"
#include "f_graph.h"

void execute()
{
    int time;
    char s[30];

    Lib_strtclk();

    /* 処理 */

    time = Lib_readclk();
    time = ( time * 9765 ) / 1000 ;

    Lib_sprintf(  s,"ELAPSED_TIME:%4d(ms)",  time );
    Lib_chrdisp( 1, 30, s );
}
```

留意事項

○ クロック単位が 9.765ms では大きい場合、クロック単位が 0.813802ms の Lib_strtclk_count をご使用ください。

Lib_readclk

機　能

実時間クロック読み取り

形　式

```
#include "f_time.h"
unsigned int Lib_readclk( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

実時間クロックを読取るもので、クロック現在値が読取れます。

戻り値

現在値を返します。

例

何らかの処理の処理時間を計測します。

```
#include "f_time.h"
#include "f_stdio.h"
#include "f_graph.h"

void execute()
{
    int time;
    char s[30];

    Lib_strtclk();

    /* 処理 */

    time = Lib_readclk();
    time = ( time * 9765 ) / 1000;

    Lib_sprintf(  s,"ELAPSED_TIME:%4d(ms)",  time );
    Lib_chrdisp( 1, 30, s );
}
```

留意事項

○ クロックの単位
　クロックの単位時間は 9.765ms で、1 単位としています。
○ 現在値が 4294967295(0xffffffff) の次は 0 になります。

Lib_strtetc

機　能

インタバルタイマ設定

形　式

```
#include "f_time.h"
void Lib_strtetc( int no, int value );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定された番号のインタバルタイマの初期設定を実施するものです。

① ***no*** はインタバルタイマ番号(0～7)です。

② ***value*** はタイマ値です。単位はミリ秒(ms)です。
[注]クロックの単位
クロックの単位時間は 9.765ms で、1単位としています。

戻り値

ありません。

例

50ms のディレイを実行します。

```
#include "f_time.h"

#define TIME_NO     0
#define TIME_VALUE  50

void time_delay( )
{
    Lib_strtetc( TIME_NO, TIME_VALUE );

    while( NO_TIME_OUT == Lib_sensetc( TIME_NO ) ) {}

    Lib_cancetc( TIME_NO );
}
```

留意事項

○ クロックの単位
クロックの単位時間は 9.765ms で、1単位としています。

Lib_sensetc

機　能

インタバルタイマ状態検知

形　式

```
#include "f_time.h"
int Lib_sensetc( int no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

インタバルタイマ状態を検知します。

① ***no*** はインタバルタイマ番号(0～7)です。

戻り値

タイマ状態を返します。

値	定　数	意　味
0x00	NO_TIME_OUT	初期状態、またはタイムアウト時間が発生していません。
0xff	TIME_OUT	タイムアウト時間が発生しました。

例

50msec のディレイを実行します。

```
#include "f_time.h"

#define TIME_NO     0
#define TIME_VALUE  50

void time_delay( )
{
    Lib_strtetc( TIME_NO, TIME_VALUE );

    while( NO_TIME_OUT == Lib_sensetc( TIME_NO ) ) {}

    Lib_cancetc( TIME_NO );
}
```

留意事項

○ クロックの単位
　クロックの単位時間は 9.765ms で、1 単位としています。

Lib_cancetc

機　能

インタバルタイマ取消

形　式

```
#include "f_time.h"
void Lib_cancetc( int no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定されたインタバルタイマを停止させ、キャンセルします。
タイマ状態は 0xff(TIME_OUT) に設定されます。

① ***no*** はインタバルタイマ番号(0～7)です。

戻り値

ありません。

例

50ms のディレイを実行します。

```
#include "f_time.h"

#define TIME_NO     0
#define TIME_VALUE  50

void time_delay( )
{
    Lib_strtetc( TIME_NO, TIME_VALUE );

    while( NO_TIME_OUT == Lib_sensetc( TIME_NO ) ) {}

    Lib_cancetc( TIME_NO );
}
```

留意事項

○ クロックの単位
　クロックの単位時間は 9.765ms で、1 単位としています。

Lib_time_delay

機　能

タイムディレイ

形　式

```
#include "f_time.h"
void Lib_time_delay( unsigned int time );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定された実時間のクロックのディレイを実行します。

① ***time*** はタイマ値です。単位はミリ秒(ms)です。

戻り値

ありません。

例

約 50ms のディレイを実行します。

```
#include "f_time.h"

#define TIME_VALUE  50

void time_delay( )
{
    Lib_time_delay( TIME_VALUE );
}
```

留意事項

○ クロックの単位時間は 9.765ms で、1 単位としています。

Lib_strtclk_count

機　能

実時間クロックスタート(微細クロックモード)

形　式

```
#include "f_time.h"
void Lib_strtclk_count( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

0.813802ms ごとにカウントされる実時間クロックをスタートするものです。
この実時間クロックは、ユーザプログラムで読取ることもできます。

戻り値

ありません。

例

何らかの処理の処理時間を計測します。

```
#include "f_time.h"
#include "f_stdio.h"
#include "f_graph.h"

void execute()
{
    int time;
    char s[30];

    Lib_strtclk_count();

    /* 処理 */

    time = Lib_readclk_count();
    time = ( time * 8138 ) / 10000;

    Lib_sprintf(  s,"ELAPSED_TIME:%4d(ms)",  time );
    Lib_chrdisp( 1, 30, s );
    Lib_stopclk_count();
}
```

留意事項

○ 最大約 53 秒までの時間が計測できます。

Lib_readclk_count

機　能

実時間クロック読み取り(微細クロックモード)

形　式

```
#include "f_time.h"
int Lib_readclk_count( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

実時間クロックを読取るもので、クロック現在値が読取れます。

戻り値

現在値を返します。

例

何らかの処理の処理時間を計測します。

```
#include "f_time.h"
#include "f_stdio.h"
#include "f_graph.h"

void execute()
{
    int time;
    char s[30];

    Lib_strtclk_count();

    /* 処理 */

    time = Lib_readclk_count();
    time = ( time * 8138 ) / 10000;

    Lib_sprintf(  s,"ELAPSED_TIME:%4d(ms)",  time );
    Lib_chrdisp( 1, 30, s );
    Lib_stopclk_count();
}
```

留意事項

○ クロックの単位
　クロックの単位時間は 0.813802ms で、1 単位としています。

Lib_time_disp

機　能

実時間クロックの読み取り表示(微細クロックモード)

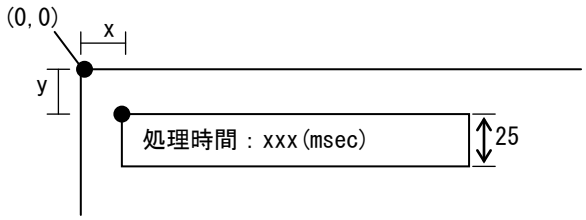
形　式

```
#include "f_time.h"
void Lib_time_disp( int xpos, int ypos );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

実時間クロックを読み取り、モニタ TV に表示します。



※ 英語モード時は ELAPSED_TIME:xxx(msec)

① ***xpos*** は表示 X 位置(0～511)です。

② ***ypos*** は表示 Y 位置(0～479)です。

戻り値

ありません。

例

何らかの処理の処理時間を計測し、表示します。

```
#include "f_time.h"
#define XPOS  1
#define YPOS  1

void execute()
{
    Lib_strtclk_count();

    /* 処理 */

    Lib_time_disp( XPOS, YPOS );
}
```

留意事項

○ Lib_stopclk_count はコールする必要はありません。

Lib_stopclk_count

機　能

実時間クロック停止(微細クロックモード)

形　式

```
#include "f_time.h"
void Lib_stopclk_count( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

微細クロックモードの実時間クロックを停止します。

戻り値

ありません。

例

何らかの処理の処理時間を計測します。

```
#include "f_time.h"
#include "f_stdio.h"
#include "f_graph.h"

void execute()
{
    int time;
    char s[30];

    Lib_strtclk_count();

    /* 処理 */

    time = Lib_readclk_count();
    time = ( time * 8138 ) / 10000;

    Lib_sprintf(  s,"ELAPSED_TIME:%4d(ms)",  time );
    Lib_chrdisp( 1, 30, s );
    Lib_stopclk_count();
}
```

留意事項

○ 通常モードの実時間クロックは停止しません。

Lib_get_time

機　能

時刻の取得

形　式

```
#include "f_calen.h"
void  Lib_get_time( TIME_T *time );
```

901	902	903	904	FVL/LNX	高分解能
	○	○	△	○	○

解　説

システムで設定されている時刻を ***time*** が指す TIME_T 構造体に格納します。

① ***time*** は構造体 TIME_T のポインタで、時刻を取得します。

構造体 TIME_T は f_calen.h 内で、次のように定義されています。

```
typedef struct stTime
{
      int  wHour;            /* 時間 (0-23)*/
      int  wMin;             /*  分  (0-59)*/
      int  wSec;             /*  秒  (0-59)*/
} TIME_T;
```

戻り値

ありません。

例

```
#include       "f_stdio.h"
#include       "f_calen.h"

void CheckTime()
{
    TIME_T dtime1,dtime2;

    Lib_get_time( &dtime1 );
    Lib_printf( "%d:%d:%d¥n",dtime1.wHour,dtime1.wMin, dtime1.wSec );
    dtime2.wHour = 10; /* 時間 */
    dtime2.wMin  = 30; /*  分  */
    dtime2.wSec  =  0; /*  秒  */
    Lib_set_time( &dtime2 );
    Lib_get_time( &dtime1 );
    Lib_printf( "%d:%d:%d¥n",dtime1.wHour,dtime1.wMin, dtime1.wSec );
}
```

留意事項

904

○ 本ライブラリは FV904p のみサポートしています。

Lib_set_time

機　能

時刻の設定

形　式

```
#include "f_calen.h"
int  Lib_set_time( TIME_T *time );
```

901	902	903	904	FVL/LNX	高分解能
	○	○	△	○	○

解　説

time が指す TIME_T 構造体の値をシステムの時刻とします。

① ****time*** は構造体 TIME_T のポインタで、時刻を指定します。

構造体 TIME_T は f_calen.h 内で、次のように定義されています。

```
typedef struct stTime
{
    int  wHour;              /* 時間 (0-23)*/
    in   twMin;              /*  分  (0-59)*/
    int  wSec;               /*  秒  (0-59)*/
} TIME_T;
```

戻り値

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

Lib_get_time の例を参照してください。

留意事項

904

○ 本ライブラリは FV904p のみサポートしています。

FVL/LNX

○ ver3.00 より、時刻設定の内部仕様が変更になりした。
Ver3.00 以前では、Lib_set_time, Lib_set_date 関数が呼ばれた際には、
設定する時刻と実際の BIOS の時刻の差分値を FVL/LNX 内部で保持し、
Lib_get_date, Lib_get_time 関数が呼ばれた際にその差分値を適用した値を返値と
していましたが、ver3.00 以降では、直接 BIOS の時刻を変更しています。

Lib_get_date

機　能

日付の取得

形　式

```
#include "f_calen.h"
void  Lib_get_date( DATE_T *date );
```

901	902	903	904	FVL/LNX	高分解能
	○	○	△	○	○

解　説

システムで設定されている日付を ***date*** が指す DATE_T 構造体に格納します。

① ****date*** は構造体 DATE_T のポインタで、日付を取得します。

構造体 DATE_T は f_calen.h 内で、次のように定義されています。

```
typedef struct stDate
{
    int     wDay;          /*  日   (1-31)*/
    int     wMonth;        /*  月   (1-12)*/
    int     wYear;         /*  年   (1980-2099)*/
    int     wDayOfWeek;    /* 曜日 (0-6) 0=日曜日*/
} DATE_T;
```

戻り値

ありません。

例

```
#include       "f_stdio.h"
#include       "f_calen.h"

void CheckDate()
{
    DATE_T ddate1,ddate2;

    Lib_get_date( &ddate1 );
    Lib_printf( "%d/%d/%d %d¥n", ddate1.wYear, ddate1.wMonth,
                                        ddate1.wDay, ddate1.wDayOfWeek );

    ddate2.wYear      = 1996; /*  年  */
    ddate2.wMonth     =   10; /*  月  */
    ddate2.wDay       =   22; /*  日  */
    ddate2.wDayOfWeek =    2; /* 曜日 */

    Lib_set_date( &ddate2 );

    Lib_get_date( &ddate1 );

    Lib_printf( "%d/%d/%d %d¥n", ddate1.wYear, ddate1.wMonth,
                                        ddate1.wDay, ddate1.wDayOfWeek );
}
```

留意事項

904

○ 本ライブラリは FV904p のみサポートしています。

Lib_set_date

機　能

日付の設定

形　式

```
#include "f_calen.h"
int  Lib_set_date( DATE_T *date );
```

901	902	903	904	FVL/LNX	高分解能
	○	○	△	○	○

解　説

date が指す DATE_T 構造体の値をシステムの日付とします。

① ****date*** は構造体 DATE_T のポインタで、日付を指定します。

構造体 DATE_T は f_calen.h 内で、次のように定義されています。

```
typedef struct stDate
{
     int      wDay;          /*  日   (1-31)*/
     int      wMonth;        /*  月   (1-12)*/
     int      wYear;         /*  年   (1980-2099)*/
     int      wDayOfWeek;    /* 曜日  (0-6)  0=日曜日*/
} DATE_T;
```

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

Lib_get_date の例を参照してください。

留意事項

903

○ 曜日設定は無視されます(内部で計算により求めています)。

904

○ 曜日設定は無視されます(内部で計算により求めています)。

○ 本ライブラリは FV904p のみサポートしています。

FVL/LNX

○ ver3.00 より、時刻設定の内部仕様が変更になりした。
Ver3.00 以前では、Lib_set_time, Lib_set_date 関数が呼ばれた際には、
設定する時刻と実際の BIOS の時刻の差分値を FVL/LNX 内部で保持し、
Lib_get_date, Lib_get_time 関数が呼ばれた際にその差分値を適用した値を返値と
していましたが、ver3.00 以降では、直接 BIOS の時刻を変更しています。

Lib_check_battery

機　能

電池の残容量確認

形　式

```
#include "f_calen.h"
int Lib_check_battery(void);
```

901	902	903	904	FVL/LNX	高分解能
		○	△		

解　説

カレンダタイマ用電池の残容量を確認します。

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常。
-1	ERROR_RETURN	電池の残容量不足。

留意事項

共通

○ バッテリー状態は電源投入時に１度だけ更新されます。

904

○ 本ライブラリは FV904p のみサポートしています。

Lib_xreadclk

機　能

実時間読み取り

形　式

```
#include "f_time.h"
void Lib_xreadclk( unsigned int *sec, unsigned int *msec, unsigned int *usec );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

実時間クロックスタートライブラリ(Lib_strtclk_count)からの経過時間を読み取ります。

① *sec* は経過時間(秒単位)へのポインタです。

② *msec* は経過時間(ミリ秒単位)へのポインタです。

③ *usec* は経過時間(マイクロ秒単位)へのポインタです。

戻り値

ありません。

例

何らかの処理の処理時間(ミリ秒単位)を計測します。

```
#include "f_time.h"

void func( void )
{
    unsigned int msec;

    Lib_strtclk_count();
    /***** 何らかの処理 *****/
    Lib_xreadclk( NULL, &msec, NULL );

}
```

留意事項

○ 出力先のポインタにNULLを設定した場合は、出力されません。

15. 数値演算ライブラリ

本ライブラリは 90X シリーズで実施している各種数値演算のうち、固定小数点形式で演算するものに限ってサービスするものです。

Lib_sinfunc

機　能

正弦関数

形　式

```
#include "f_math.h"
int Lib_sinfunc( int angle );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

角度から sine を計算するもので、1.0 を 32767 とした -32767～32767 を返答します。

① ***angle*** は角度 10 倍値です。

戻り値

-32767～32767 の数値の形で返します。
ただし、1.0 を 32767 にスケーリングした値です。
負数は 2 補数値とします。

例

30 度の sin を求めます。

```
#include "f_math.h"
#define ANGLE 30

double want_sin()
{
    int sindata;
    double sin;

    sindata = Lib_sinfunc( ANGLE * 10 );
    sin = sindata / 32767;
    return( sin );
}
```

留意事項

ありません。

Lib_cosfunc

機　能

余弦関数

形　式

```
#include "f_math.h"
int Lib_cosfunc( int angle );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

角度から cosine を計算するもので、1.0 を 32767 とした -32767～32767 を返答します。

① ***angle*** は角度 10 倍値です。

戻り値

-32767～32767 の数値の形で返します。
ただし、1.0 を 32767 にスケーリングした値です。
負数は 2 補数値とします。

例

30 度の cos を求めます。

```
#include "f_math.h"
#define ANGLE 30

double want_cos()
{
    int cosdata;
    double cos;

    cosdata = Lib_cosfunc( ANGLE * 10 );
    cos = cosdata / 32767;
    return( cos );
}
```

留意事項

ありません。

Lib_sqrt32

機　能

平方根関数

形　式

```
#include "f_math.h"
int Lib_sqrt32( int value );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

32 ビット整数値の平方根を 0～65535 のワード値として返答します。
32 ビット整数値が負数(すなわち、先頭ビットがオン)の値については、ゼロを返答します。

① ***value*** は数値です。

戻り値

計算結果を返します。

例

(25, 40) - (225, 300)の距離を求めます。

```
#include "f_math.h"
#define  XS_POS  25
#define  YS_POS  40
#define  XE_POS  225
#define  YE_POS  300

int want()
{
    int lw;
    int xd,yd;

    xd=XE_POS-XS_POS;
    yd=YE_POS-YS_POS;
    lw=xd*xd+yd*yd;
    lw = Lib_sqrt32( lw );
    return( lw );
}
```

留意事項

ありません。

Lib_atan360

機　能

逆正接関数

形　式

```
#include "f_math.h"
int Lib_atan360( int x, int y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

y/x のアークタンジェントを計算し、0.0 から 359.9 度の範囲の値を返します。
テーブル索引方式ですので高速に計算ができます。

① *x* の範囲は -2147483648 から 2147483647 です。

② *y* の範囲は -2147483648 から 2147483647 です。

戻り値

最大誤差 ±0.1 度の角度値 0.0 から 359.9 度の値の 10 倍値を返します。

例

X が 90, Y が 15 のアークタンジェントを計算し、結果を表示します。

```
#include "f_stdio.h"
#include "f_graph.h"
#include "f_math.h"

void atan360( void )
{
    char s[ 50 ];
    short result;

    result = Lib_atan360( 9015 );
    Lib_sprintf( s, "15 / 90 : %d", result );
    Lib_chrdisp( 10, 20, s );
}
```

留意事項

ありません。

16. 特殊計測・計算処理ライブラリ

本ライブラリは特殊計測計算処理に関するものです。

Lib_calcline

機　能

最小自乗法直線近似

形　式

```
#include  "f_math.h"
int  Lib_calcline( int point_n, POINT *cood, LINE *coeff );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

与えられた座標値群から、最小自乗法により直線 ax＋by＋c＝0 の各係数値 a, b, c を決定するものです。
なお、本ライブラリにて採用する座標点は、TV 座標に限定せず、広く一般の直交座標系のものにも適用できます。

① ***point_n*** は直線近似する座標の点数を 2 から 1000 までの数値にて指定します。

② ****cood*** は座標値群へのポインタで、(座標点数)で与えた点数分の、直線近似する座標点群を与えます。座標点は TV 座標上の点のみならず、任意の(x,y)座標点が与えられ、-32768 から +32767 の範囲の数値が与えられます。従って、所望する精度に応じて、0.1 画素を単位とした直線近似式なども、求めることが出来ます。

座標値群（ポインタ）→

	バイト
座標点 1 の X 位置(=x1)	+00
座標点 1 の Y 位置(=y1)	+04
座標点 2 の X 位置(=x2)	+08
座標点 2 の Y 位置(=y2)	+12
・ ・	
座標点 n の X 位置(=xn)	+8n-8
座標点 n の Y 位置(=yn)	+8n-4

③ ****coeff*** は直線係数へのポインタで、近似された直線 ax＋by＋c＝0 の係数 a,b は 65536 倍した 32 ビット整数値の形で、係数 c は 16 倍した 32 ビット整数値の形にて得られます。

直線係数（ポインタ）→

	バイト
係数 a を 65536 倍した 32 ビット整数値	+00
係数 b を 65536 倍した 32 ビット整数値	+04
係数 c を 16 倍した 32 ビット整数値	+08

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	CALC_IMPOSSIBLE	a,b の係数値が不定となるような座標データが与えられました。(与えられた座標値がすべて同一の値である場合などに、このようになります。)
-2	CALC_OVERFLOWED	a,b,c の係数のうちのいずれかが 32 ビットをオーバフローしました。(通常は発生しません。)

例

```
#include "f_math.h"
/* Lib_calcline()の帰り値 */

#define  RANGE_A        65536          /* 直線係数 a の倍率値 */
#define  RANGE_B        65536          /* 直線係数 b の倍率値 */
#define  RANGE_C        16             /* 直線係数 c の倍率値 */
#define  CHAR_LENGTH    100            /* 文字列バッファ長       */

#define  X_DISP         1              /* 表示桁              */
#define  Y_DISP         1              /* 表示行              */

#define  EPS            2.0e-3         /* ε                   */



void  main()
{
    static POINT   point_data[4];      /* 座標データ格納バッファ */
    LINE    coeff;
    char    s[CHAR_LENGTH];

    point_data[0].x = 6;
    point_data[0].y = 6;
    point_data[1].x = 10;
    point_data[1].y = 13;
    point_data[2].x = 15;
    point_data[2].y = 12;
    point_data[3].x = 19;
    point_data[3].y = 17;

    switch( Lib_calcline( 4, point_data, &coeff ) )
    {
        case  NORMAL_RETURN:           /* 正常終了 */
            Lib_sprintf( s, "a =%10d  b =%10d  c =%10d¥n¥r", coeff.a,
                                                    coeff.b, coeff.c );
            Lib_chrdisp( X_DISP, Y_DISP + 8, s );
            break;

        case  CALC_IMPOSSIBLE:         /* 計算不能 */
            Lib_chrdisp( X_DISP, Y_DISP + 6, "計算出来ませんでした" );
            break;

        case  CALC_OVERFLOWED:         /* 数値のオーバーフロー */
            Lib_chrdisp( X_DISP, Y_DISP + 6, "数値がオーバーフローしました" );
            break;
    }
}
```

留意事項

ありません。

Lib_fcalcircl

機　能

最小自乗法円近似

形　式

```
#include  "f_math.h"
int  Lib_fcalcircl( int point_n, POINT_T *cood , double *a,
                                                  double *b, double *r );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

与えられた座標値群から、最小自乗法により円 (x-a)2 + (y-b)2 = r2 の各係数値ａ， ｂ，ｒを決定するものです。

① ***point_n*** は円近似する座標の点数を 3 から 16 までの数値にて指定します。

② ****cood*** は座標値群へのポインタで、「座標点数」で与えた点数分の、円近似する座標点群を与えます。
ただし、座標値は X, Y ともに 0～511 までのものとします。

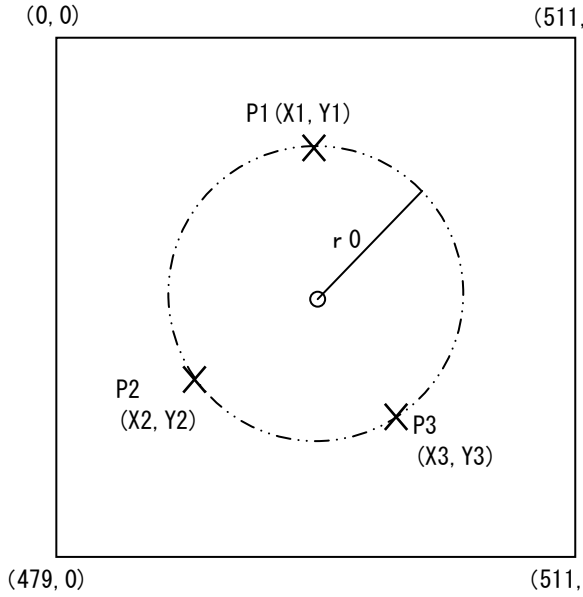
座標値群(ポインタ)→

	バイト
座標点 1 の X 位置(=x1)	+00
座標点 1 の Y 位置(=y1)	+04
座標点 2 の X 位置(=x2)	+08
座標点 2 の Y 位置(=y2)	+12
▪ ▪	
座標点 n の X 位置(=xn)	+8n-8
座標点 n の Y 位置(=yn)	+8n-4

③ ****a*** は円の中心 X 座標へのポインタを設定します。

④ ****b*** は円の中心 Y 座標へのポインタを設定します。

⑤ ****r*** は円の半径へのポインタを設定します。



P 点(P1,P2,P3・・・・・Pn)より X0,Y0 および r0 を求める

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	CALC_IMPOSSIBLE_X	a の係数値が不定となるような座標データが与えられました。
-2	CALC_OVERFLOWED_Y	b の係数値が不定となるような座標データが与えられました。
-3	ILLEGAL_POINT_N	座標点数が 3～16 の範囲に入っていません。

例

```
#include      "f_math.h"

typedef struct
{
    int  x;
    int  y;
} POINT_T;

#define  POINT_N   16
void  main()
{
    int      i, time, error;
    double   xo, yo, r;
    POINT_T  point[POINT_N];

    point [0].x = 448;  point [0].y = 328;
    point [1].x = 469;  point [1].y = 333;
    point [2].x = 487;  point [2].y = 349;
    point [3].x = 488;  point [3].y = 355;
    point [4].x = 492;  point [4].y = 365;
    point [5].x = 487;  point [5].y = 391;
    point [6].x = 478;  point [6].y = 402;
    point [7].x = 468;  point [7].y = 408;
    point [8].x = 453;  point [8].y = 412;
    point [9].x = 431;  point [9].y = 407;
    point[10].x = 414;  point[10].y = 391;
    point[11].x = 412;  point[11].y = 387;
    point[12].x = 409;  point[12].y = 377;
    point[13].x = 414;  point[13].y = 350;
    point[14].x = 423;  point[14].y = 339;
    point[15].x = 432;  point[15].y = 333;

    for( i = 0; i < POINT_N; i++ )  Lib_pset( point[i].x, point[i].y, 1 );

    if( NORMAL_RETURN == ( error = Lib_fcalcircl
                        ( POINT_N, point, &xo, &yo, &r ) ) )
    {
        Lib_printf( "xo = %lf  yo = %lf  r = %lf¥n¥r", xo, yo, r );

        Lib_drawcircle( ( int )xo, ( int )yo, ( int )r );
    }
    else  Lib_printf( "error = %d¥n¥r", error );
    Lib_time_delay(5000);
}
```

留意事項

ありません。

Lib_fcalelps

機　能

最小自乗法楕円近似

形　式

```
#include "f_math.h"
int  Lib_fcalelps( int point_n, POINT *coord, double *xc, double *yc,
                   double *rad_long, double *rad_short, double *angle );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	

解　説

与えられた座標値群から、最小自乗法により楕円の各係数値を決定するものです。
各係数値と楕円の関係は下図を御覧ください。

① ***point_n*** は楕円近似する座標の点数を 5 から 32 までの数値にて指定します。

② ****coord*** は座標値群へのポインタで、「座標点数 point_n 」で与えた点数分の、楕円近似する座標点群を与えます。ただし、座標値は X,Y ともに 0～511 までのものとします。

座標値群(ポインタ)→	バイト
座標点 1 の X 位置(=x1)	+00
座標点 1 の Y 位置(=y1)	+04
座標点 2 の X 位置(=x2)	+08
座標点 2 の Y 位置(=y2)	+12
・	
・	
座標点 n の X 位置(=xn)	+8n-8
座標点 n の Y 位置(=yn)	+8n-4

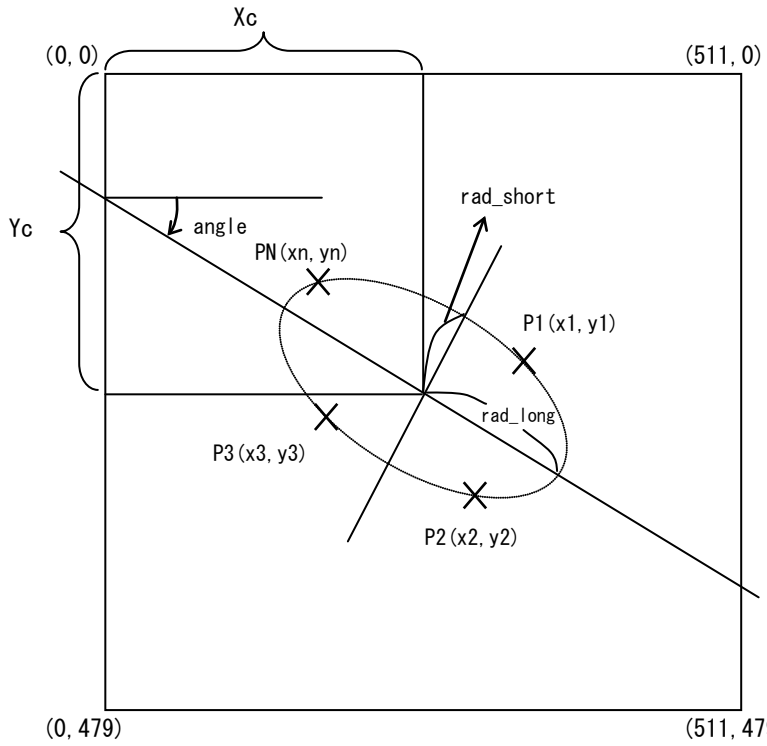
③ ****xc*** は楕円の中心 X 座標へのポインタを設定します。

④ ****yc*** は楕円の中心 Y 座標へのポインタを設定します。

⑤ ****rad_long*** は主軸半径へのポインタを設定します。

⑥ ****rad_short*** は副軸半径へのポインタを設定します。

⑦ ***angle*** は楕円の主軸傾斜角へのポインタを設定します。単位は「度」です。



N 点(P1,P2,P3・・・・PN)より
xc,yc,rad_long,rad_short,angle を
求める

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ELP_ILLEGAL_POINT_N	座標点数が不適当です。
-2	ELP_CALC_IMPOSSIBLE	与えられた座標では計算できません。
-3	ELP_NOT_ENOUGH_WORK	ワークメモリが足りません。

例

```
#include  "f_math.h"
double sin( double );
double cos( double );
void  main()
{
    int      i;
    POINT    point[32];
    double   xc, yc, rad_long, rad_short, angle;
    int      sin_val, cos_val;
    point[ 0].x = 310;    point[ 0].y = 341;
    point[ 1].x = 304;    point[ 1].y = 338;
    point[ 2].x = 296;    point[ 2].y = 332;
    point[ 3].x = 288;    point[ 3].y = 323;
    point[ 4].x = 278;    point[ 4].y = 311;
    point[ 5].x = 268;    point[ 5].y = 298;
    point[ 6].x = 258;    point[ 6].y = 283;
    point[ 7].x = 248;    point[ 7].y = 267;
    point[ 8].x = 239;    point[ 8].y = 250;
    point[ 9].x = 231;    point[ 9].y = 234;
    point[10].x = 224;    point[10].y = 218;
    point[11].x = 219;    point[11].y = 204;
    point[12].x = 216;    point[12].y = 192;
    point[13].x = 214;    point[13].y = 183;
    point[14].x = 215;    point[14].y = 176;
    point[15].x = 217;    point[15].y = 172;
    point[16].x = 221;    point[16].y = 172;
    point[17].x = 227;    point[17].y = 175;
    point[18].x = 235;    point[18].y = 181;
    point[19].x = 243;    point[19].y = 190;
    point[20].x = 253;    point[20].y = 202;
    point[21].x = 263;    point[21].y = 215;
    point[22].x = 273;    point[22].y = 230;
    point[23].x = 283;    point[23].y = 246;
    point[24].x = 292;    point[24].y = 263;
    point[25].x = 300;    point[25].y = 279;
    point[26].x = 307;    point[26].y = 295;
    point[27].x = 312;    point[27].y = 309;
    point[28].x = 315;    point[28].y = 321;
    point[29].x = 317;    point[29].y = 330;
    point[30].x = 316;    point[30].y = 337;
    point[31].x = 314;    point[31].y = 341;
    for( i = 0; i < 32; i++ )  Lib_pset( point[i].x, point[i].y, 1 );
    Lib_fcalelps( 32, point, &xc, &yc, &rad_long, &rad_short, &angle );
    angle = angle * 3.1415926 / 180.0;
    sin_val = ( int )( sin( angle ) * 1024.0 );
    cos_val = ( int )( cos( angle ) * 1024.0 );
    Lib_time_delay( 1000 );
    Lib_circle( ( int )xc, ( int )yc, ( int )rad_long,
                        ( int )rad_short, sin_val, cos_val, GRAPH_DRAW );
    Lib_time_delay( 3000 );
}
```

留意事項

○ 例の POINT は "f_math.h" に定義されている構造体です。

Lib_fcalquad

機　能

最小自乗法 2 次曲線近似

形　式

```
#include  "f_math.h"
int  Lib_fcalquad( int point_n, DPNT_T *point, double *a,
                                                double *b, double *c );
```

901	902	903	904	FVL/LNX	高分解能
	○	○	○	○	△

解　説

与えられた座標値群から、最小自乗法により 2 次曲線 y=ax2+bx+c の各係数値 a,b,c を決定するものです。
なお、本ライブラリにて採用する座標点は TV 座標に限定せず、広く一般の直交座標系のものにも適用できます。

① ***point_n*** は 2 次曲線近似する座標の点数を 3 から 512 までの数値にて指定します。
② ****point*** は座標値群へのポインタで、「座標点数」で与えた点数分の 2 次曲線近似する座標点群を与えます。
構造体 DPNT_T は f_math.h 内で以下のように定義されています。

```
typedef struct
{
    double  x;
    double  y;
}   DPNT_T;
```

③ ****a*** は y=ax2+bx+c における係数 a です。
④ ****b*** は y=ax2+bx+c 似おける係数 b です。
⑤ ****c*** は y=ax2+bx+c 似おける係数 c です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	QUAD_ILLEGAL_POINT_N	座標点数が不適当です。
-2	QUAD_CALC_IMPOSSIBLE	与えられた座標では計算不可能です。
-3	QUAD_NOT_ENOUGH_WORK	ワークメモリが足りません。

例

ファイル quadsmpl.c として csc90X¥sample(fv904¥sample)にインストールされていますので、そちらを参照してください。

留意事項

○ 高分解能使用時は画像サイズが最大 1280×1030 画素程度で使用してください。
それ以上の画像サイズの場合はオーバーフローする可能性があります。

Lib_drawquad

機　能

2 次曲線描画

形　式

```
#include  "f_graph.h"
int  Lib_drawquad( int xs, int ys, int xe, int ye, double a,
                                                  double b, double c );
```

901	902	903	904	FVL/LNX	高分解能
	○	○	○	○	

解　説

線画フレームバッファの表示対象領域内に与えられた 2 次曲線 y=ax2+bx+c を描画します。

① ***xs*** は表示対象領域の X 始端位置です。
② ***ys*** は表示対象領域の Y 始端位置です。
③ ***xe*** は表示対象領域の X 終端位置です。
④ ***ye*** は表示対象領域の Y 終端位置です。
⑤ ***a*** は y＝ax2＋bx＋c における係数 a です。
⑥ ***b*** は y＝ax2＋bx＋c における係数 b です。
⑦ ***c*** は y＝ax2＋bx＋c における係数 c です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	引数が不適当です。

例

ファイル quadsmpl.c として csc90X¥sample(fv904¥sample)にインストールされていますので、そちらを参照してください。

留意事項

ありません。

17. アフィン変換ライブラリ

Lib_rotation

機　能

画像の回転

形　式

```
#include "f_affine.h"
int Lib_rotation ( int src_mem_no, int dst_mem_no, int theta, int xc, int yc );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリの画像を最近傍法により回転し、ステージパラメータで規定される「処理領域」で囲まれる結果を濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号. です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号. です。
src_mem_no とは、異なるメモリ番号を指定してください。

③ ***theta*** は回転角です。
時計回りの回転角の 10 倍ちを与えます。指定範囲は 0 から 3600 です。

④ ***xc*** は回転中心 X 方向位置です。

⑤ ***yc*** は回転中心 Y 方向位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し(255,239)を中心に 30 度回転します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void rotation( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_rotation( CURRENT_MEMORY, mem_no, 300, 255, 239 );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 本ライブラリの濃度値補間は最近傍法を用いています。

Lib_xrotation

機　能

画像の回転Ⅱ

形　式

```
#include "f_affine.h"
int Lib_xrotation ( int src_mem_no, int dst_mem_no, int theta, int xc, int yc );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリの画像を線形補間法により回転し、ステージパラメータで規定される「処理領域」で囲まれる結果を濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号. です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号. です。
src_mem_no とは、異なるメモリ番号を指定してください。

③ ***theta*** は回転角です。
時計回りの回転角の 10 倍値を与えます。指定範囲は 0 から 3600 です。

④ ***xc*** は回転中心 X 方向位置です。

⑤ ***yc*** は回転中心 Y 方向位置です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し(255,239)を中心に 30 度回転します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void rotation( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_xrotation( CURRENT_MEMORY, mem_no, 300, 255, 239 );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 本ライブラリの濃度値補間は線形補間法を用いています。

Lib_scale

機　能

画像の拡大縮小

形　式

```
#include "f_affine.h"
int Lib_scale( int src_mem_no, int dst_mem_no, double sx, double sy );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

処理対象濃淡画像メモリの座標(0,0)の位置を基準として画像を最近傍法により拡大縮小し、ステージパラメータで規定される「処理領域」で囲まれる範囲の画像を結果格納用濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号.です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号.です。
src_mem_no とは、異なるメモリ番号を指定してください。

③ ***sx*** はＸ方向拡大率です。
☆拡大時の指定範囲は　1000 ≧ sx ≧ 1　です。
☆縮小時の指定範囲は　1 ≧ sx ＞ 0　です。

④ ***sy*** はＹ方向拡大率です。
☆拡大時の指定範囲は　1000 ≧ sy ≧ 1　です。
☆縮小時の指定範囲は　1 ≧ sy ＞ 0　です。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し 1.5 倍に拡大します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void scale( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_scale( CURRENT_MEMORY, mem_no, 1.5, 1.5 );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 本ライブラリの濃度値補間は最近傍法を用いています。

機　能

画像の拡大縮小Ⅱ

形　式

```
#include "f_affine.h"
int Lib_xscale( int src_mem_no, int dst_mem_no, double sx, double sy );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

処理対象濃淡画像メモリの座標(0,0)の位置を基準として画像を線形補間法により拡大縮小し、ステージパラメータで規定される「処理領域」で囲まれる範囲の画像を結果格納用濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号.です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号.です。
src_mem_no とは、異なるメモリ番号を指定してください。

③ ***sx*** はＸ方向拡大率です。
☆拡大時の指定範囲は　1000 ≧ sx ≧ 1　です。
☆縮小時の指定範囲は　1 ≧ sx ＞ 0　です。

④ ***sy*** はＹ方向拡大率です。
☆拡大時の指定範囲は　1000 ≧ sy ≧ 1　です。
☆縮小時の指定範囲は　1 ≧ sy ＞ 0　です。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し 1.5 倍に拡大します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void scale( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_xscale( CURRENT_MEMORY, mem_no, 1.5, 1.5 );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ 本ライブラリの濃度値補間は線形補間法を用いています。

Lib_enlarge

機　能

画像の 2 倍拡大

形　式

```
#include "f_affine.h"
int Lib_enlarge( int src_mem_no, int dst_mem_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

処理対象濃淡画像メモリの座標(0,0)の位置を基準として画像を 2 倍に拡大し、ステージパラメータで規定される「処理領域」で囲まれる範囲の画像を結果格納用濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号. です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号. です。
src_mem_no とは、異なるメモリ番号を指定してください。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し 2 倍に拡大します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void enlarge( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_enlarge( CURRENT_MEMORY, mem_no );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
 }
```

留意事項

○ Lib_scale に比べて、拡大率固定のため、高速に拡大できます。
○ 拡大は面積比で 4 倍になります。

Lib_reduce

機　能

画像の 1/2 縮小

形　式

```
#include "f_affine.h"
int Lib_reduce( int src_mem_no, int dst_mem_no );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

処理対象濃淡画像メモリの座標(0,0)の位置を基準として画像を 1/2 に縮小し、ステージパラメータで規定される「処理領域」で囲まれる範囲の画像を結果格納用濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号.です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号.です。
src_mem_no とは、異なるメモリ番号を指定してください。

戻り値

処理結果

値	定　数	意　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し 1/2 倍に縮小します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void reduce( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_reduce( CURRENT_MEMORY, mem_no );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

○ Lib_scale に比べて、縮小率固定のため、高速に縮小できます。
○ 縮小は面積比で 1/4 になります。

Lib_shift

機　能

画像の平行移動

形　式

```
#include "f_affine.h"
int Lib_shift( int src_mem_no, int dst_mem_no, int dx, int dy );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

濃淡画像メモリの画像を平行移動し、ステージパラメータで規定される「処理領域」で囲まれる結果を濃淡画像メモリに格納します。

① ***src_mem_no*** は処理対象濃淡画像メモリ番号. です。

② ***dst_mem_no*** は結果格納用濃淡画像メモリ番号. です。
src_mem_no とは、異なるメモリ番号を指定してください。

③ ***dx*** は X 方向移動量です。
プラスで左方向に、マイナスで右方向に移動します。

④ ***dy*** は Y 方向移動量です。
プラスで上方向に、マイナスで下方向に移動します。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

カレント濃淡画像メモリに画像を入力し左に 50 画素移動します。

```
#include "f_stdlib.h"
#include "f_video.h"
#include "f_affine.h"

void shift( void )
{
    int mem_no;

    Lib_display_control( GRAY_PLANE | LINE_PLANE | CHAR_PLANE );
    Lib_input_video_control( GRAY_PLANE );
    Lib_freeze( TRANSMIT );
    mem_no = Lib_alloc_gray_memory();

    Lib_shift( CURRENT_MEMORY, mem_no, 50, 50 );

    Lib_change_gray_memory( mem_no );
    Lib_video_transmit( GRAY_PLANE );
}
```

留意事項

ありません。

Lib_pol_to_rec

機　能

扇型エリアを矩形エリアに変換

形　式

```
#include "f_affine.h"
int Lib_pol_to_rec( int src_mem_no int dst_mem_no, int xc, int yc, int rs,
                    double qs, int re, double qe, int write_xs, int write_ys,
                                                  int *x_offset, int *y_offset );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

入力メモリ上の扇型エリアを矩形エリアに変換して出力メモリへ書き込み,矩形エリアの縦横サイズを得ます。

① ***src_mem_no*** は扇型エリアの濃淡画像メモリ番号.を指定します。

② ***dst_mem_no*** は矩形エリアを書き込む濃淡画像メモリ番号.を指定します。
src_mem_no と同じメモリ番号.は指定できません。

③ ***xc*** は円弧中心 X 座標を指定します。

④ ***yc*** は円弧中心 Y 座標を指定します。

⑤ ***rs*** は扇型エリアの領域の始点半径を画素単位で指定します。
（始点半径 ＞ 終点半径としてください）

⑥ ***qs*** は扇型エリアの領域の始点角度です。
時計の 3 時方向を 0.0 とした時計回りで指定します。(0.0..360.0)

⑦ ***re*** は扇型エリアの領域の終点半径を画素単位で指定します。
（始点半径 ＞ 終点半径としてください）

⑧ ***qe*** は扇型エリアの領域の終点角度です。
時計の 3 時方向を 0.0 とした時計回りで指定します。(0.0..360.0)

⑨ ***write_xs*** は矩形エリアの領域の始点 X 座標を指定します。

⑩ ***write_ys*** は矩形エリアの領域の始点 Y 座標を指定します。

⑪ ****x_offset*** は矩形エリアの領域の X オフセットを返します。

⑫ ****y_offset*** は矩形エリアの領域の Y オフセットを返します。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	パラメータエラーです。
-2	なし	終点角度まで処理できませんでした。

例

濃淡画像メモリ０上の円弧中心 (256, 240)、始点半径 150, 終点半径 50, 始点角度 45.0, 終点角度 270.0 で示される円弧エリアを濃淡画像メモリ１の座標 (100, 100)を始点とした矩形エリアに変換します。

```
#include "f_affine.h"

void polar_to_rectangular()
{
   int x_offset, y_offset;

   Lib_pol_to_rec( 0, 1, 255, 240, 150, 45.0, 50, 270.0, 100,
                                          100, &x_offset, &y_offset );
}
```

留意事項

○ 戻り値が-2 の場合、処理可能な部分については矩形エリアの書き込みを行います。

18. 座標変換(キャリブレーション)ライブラリ

モニタ TV の座標系で表される(x,y)を、ロボットの座標系のような実際的な「もの」の座標系(X,Y)に変換するための、変換係数を決めたり、実際の座標変換の作業を実施するライブラリです。

18.1　前提条件

①平面変換

変換は、(x,y)平面上の点のみとし、z 座標、すなわち、TV カメラの光軸と並行な座標については何ら変換の対象とせず、専ら、垂直な平面についてのみ変換するものです。

②元座標と変換先座標系との平行性

＜変換式＞

X, Y 座標系が右手系のとき：

```
X=a-bx+cy
Y=d+cx+by
```

X, Y 座標系が左手系、つまり、TV 座標系と同じ向きのとき：

```
X=a-bx-cy
Y=d+cx-by
```

右手系：X 軸を右方向とした時 Y 軸が上方向になるような通常の座標系をいいます。
左手系：TV 座標系のように X 軸を右方向とした時 Y 軸が下方向となるような座標系をいいます。

上記に見られるように、元の座標と変換先の座標とは完全に平行である、つまり、変換は、回転・スケーリング・並進により達成できるものとします。

$$U=U0+S\times\exp(\iota\theta)\times v$$

ここで、$U=X+\iota Y$
$U0=X0+\iota Y0$(並進成分)
S はスケーリングファクタ
θ 派回転角
$v=x+\iota y$

前記の式との対応関係でいえば下記のようになります。

$a=X_0$
$d=Y_0$
$b=S\times\cos(\theta)$
$c=S\times\sin(\theta)$

留意事項　座標変換(キャリブレーション)ライブラリは、TV 座標系からロボット座標系への変換のみならず、もっと広い範囲に使用できます。
ただし、その際には、下記事項に注意する必要があります。

①元座標は左手系
元座標は TV 座標系を想定していますので左手座標系、つまり、x 座標を右方向にとると y 座標は下方向となるようような座標系に固定されています。
一方、変換後の座標系は左手座標系でも、右座標系でも構いません。

②元座標として取り得る範囲は、-32768 から +32767 の範囲です。
変換後座標としては、-2147483648 から +2147483647 の範囲です。

Lib_calibstart

機　能

キャリブレーションの開始

形　式

```
#include "f_calib.h"
void Lib_calibstart( int type );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

(x,y)座標系から(X,Y)座標系への、変換係数を決定するための一連の操作を開始します。本ライブラリを呼び出し後、複数回(2 回以上) Lib_calibposition を呼び出し、最後に Lib_calibfinish を呼び出すことで、変換係数の計算が終了します。

① ***type*** は座標系タイプを指定します。

値	定　数	意　味
0	RIGHT_HAND	右手系を指定します。
1	LEFT_HAND	左手系を指定します。

(x,y)座標系は、TV 座標系を想定しているため左手系の座標系に固定してあります。
(X,Y)座標系は、左手系でも右手系でも構わず、type で指定した座標系になります。

戻り値

ありません。

例

変換係数を決定し、元座標（150, 200）の変換後の座標をもとめます。

```
#include "f_calib.h"

void sample( void )
{
    struct _coefficients coefficients;
    struct _original original;
     struct _conv conv;

    Lib_calibstart( LEFT_HAND );

    Lib_calibposition( 10, 20, 100, 200 );
    Lib_calibposition( 500, 400, 5000, 4000 );

    Lib_calibfinish( &coefficients );

    original.x = 150;
    original.y = 200;
    Lib_coordconv( &coefficients, &original, &conv );

        /* 何らかの処理 */

}
```

留意事項

○ 左手系とは、TV 座標系のように、x 軸を右方向としたとき y 軸方向が下方向となるような座標です。
右手系とは、x 軸を右方向としたとき y 軸が上方向となるような座標系です。

Lib_calibposition

機　能

対応点位置設定

形　式

#include "f_calib.h"
void Lib_calibposition(int ***x***, int ***y***, int ***X***, int ***Y***);

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

元の座標系の座標点(x,y)と、その点が対応する変換後座標系での座標位置(X, Y)を与え、ティーチングを実施します。
ティーチングは異なる座標点を複数回(2回以上)与えます。
つまり、連続して本ライブラリを2回以上呼び出す必要があります。

① ***x*** は元座標X位置を -32768 から 32767 の範囲で指定します。

② ***y*** は元座標Y位置を -32768 から 32767 の範囲で指定します。

③ ***X*** は変換後座標X位置を -2147483648 から 2147483647 の範囲で指定します。

④ ***Y*** は変換後座標Y位置を -2147483648 から 2147483647 の範囲で指定します。

x, yはふつうTV座標値そのものを与えます。
小数点以下の精度がある場合は、座標値を整数倍(例えば10倍、100倍)したものを与えます。TV座標以外のものを与える場合には、左手系であることに注意してください。
X, Yは Lib_calibstart で指定した座標系タイプにしたがって座標値を与えてください。
異なる座標系を使用した場合、変換係数が正しく求められません。
与える座標値は求める精度に応じて整数倍したものを与えます。

戻り値

ありません。

例

Lib_calibstart の例を参照してください。

留意事項

○ 本ライブラリの呼び出し前に必ず Lib_calibstart を呼び出しておいてください。
○ 3回以上本ライブラリを呼び出した場合は最小自乗法で変換係数が計算されます。
できるだけ多くの座標を与えて計測誤差を補正してください。

Lib_calibfinish

機　能

キャリブレーションの終了

形　式

```
#include "f_calib.h"
int Lib_calibfinish( struct _coefficients  *coefficients );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

一連のキャリブレーション操作の終了を通知します。

① ***coefficients*** は構造体 _coefficients のポインタです。

座標変換係数は、coefficients が指す構造体 _coefficients 内に格納します。
構造体 _coefficients は f_calib.h 内で次のように宣言されています。

```
struct _coefficients
{
    int    npoints;       /* 使用座標点数 */
    int    cord_type;     /* 座標系タイプ */
    long   X_offset;      /* X オフセット */              (注)
    long   Y_offset;      /* Y オフセット */              (注)
    long   scaling;       /* 座標スケーリングファクタ */  (注)
    int    rotate_angle;  /* 回転角 */
};
```

(注)：FVL/LNX をお使いの場合は int 型となります

使用座標点数 ····· 座標変換の計算に使用した座標対応点の数です。
Lib_calibposition の呼び出し回数に相当します。
数値範囲は 2 から 32767 です。

座標系タイプ ····· Lib_calibstart で指定した座標系タイプが格納されます。
0:右手系　　1:左手系

X オフセット····· 変換後座標 X のオフセット値を 16 倍したものが格納されます。

Y オフセット····· 変換後座標 Y のオフセット値を 16 倍したものが格納されます。
座標スケーリングファクタ··スケーリングファクタを 256 倍して整数値化したものです。

回転角 ····· X 軸方向を 0 度とした時計回りの回転角の 10 倍値を格納します。
数値範囲は 0 から 3600 です。

戻り値

処理結果

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

Lib_calibstart の例を参照してください。

留意事項

○ 本ライブラリの呼び出し前に必ず Lib_calibstart, Lib_calibposition を呼び出しておいてください。
Lib_calibposition は 2 回以上呼び出しておく必要があります。

○ 異常終了は使用座標点数が 2 未満、あるいは内部演算でオーバーフローが発生したことを通知します。

Lib_coordconv

機　能

座標変換実施

形　式

```
#include "f_calib.h"
int Lib_coordconv(  struct _coefficients *coefficients,
                            struct _original *original, struct _conv *conv );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

指定された座標変換係数を使用し、次の計算式で座標変換を行います。

右手系

$X=Xo+S(-x\times\cos(\theta)+y\times\sin(\theta))$

$Y=Yo+S(\ x\times\sin(\theta)+y\times\cos(\theta))$

左手系

$X=Xo+S(-x\times\cos(\theta)-y\times\sin(\theta))$

$Y=Yo+S(\ x\times\sin(\theta)-y\times\cos(\theta))$

Xo：X オフセット
Yo：Y オフセット
S　：座標スケーリングファクター
θ　：回転角

① ***coefficients*** は構造体 _coefficients のポインタで、座標変換係数を指定します。
構造体 _coefficients は f_calib.h 内で次のように宣言されています。

```
struct _coefficients
{
    int    npoints;       /* 使用座標点数 */
    int    cord_type;     /* 座標系タイプ */
    long   X_offset;      /* X オフセット */              (注)
    long   Y_offset;      /* Y オフセット */              (注)
    long   scaling;       /* 座標スケーリングファクタ */  (注)
    int    rotate_angle;  /* 回転角 */
};
```

(注):FVL/LNX をお使いの場合は int 型となります。

使用座標点数 ····· 座標変換の計算に使用した座標対応点の数です。
Lib_calibposition の呼び出し回数に相当します。
数値範囲は 2 から 32767 です。
座標系タイプ ····· Lib_calibstart で指定した座標系タイプが格納されます。
0：右手系　　1：左手系
X オフセット····· 変換後座標 X のオフセット値を 16 倍したものが格納されます。
Y オフセット····· 変換後座標 Y のオフセット値を 16 倍したものが格納されます。
座標スケーリングファクタ··スケーリングファクタを 256 倍して整数値化したものです。
回転角 ····· X 軸方向を 0 度とした時計回りの回転角の 10 倍値を格納します。
数値範囲は 0 から 3600 です。

② ***original*** は構造体 _original のポインタで、元座標を指定します。
元座標は TV 座標系に合わせて左手系で指定します。

構造体 _original は "f_calib.h" 内で次のように宣言されています。

```
struct _original
{
    short    x;    /* 元座標 x */
    short    y;    /* 元座標 y */
};
```

③ ***conv*** は構造体 _conv のポインタで、変換後座標が格納されます。
構造体 _conv は "f_calib.h" 内で次のように宣言されています。

```
struct _conv
{
    long    x;    /* 変換後座標 X */        (注)
    long    y;    /* 変換後座標 Y */        (注)
};
```

(注) : FVL/LNX をお使いの場合は int 型となります。

戻り値

処理結果

値	定数	意味
0	NORMAL_RETURN	正常終了しました。
-1	ERROR_RETURN	異常終了しました。

例

Lib_calibstart の例を参照してください。

留意事項

○ 異常終了は使用座標点数が 2 未満、あるいは座標系タイプが 0,1 以外の場合です。

Lib_xcalibstart

機　能

拡張キャリブレーションの開始

形　式

```
#include   "f_calib.h"
int  Lib_xcalibstart( int  order, int  data_n );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

ある座標系上の点(x、y)から、その座標系とは異なる座標系(X,Y)への変換を行う際の、座標変換係数を決定する一連の操作を開始します。
変換係数を求める際に必要となる座標の事を基準点といいます。
本ライブラリを呼び出し後、複数回(最大でも項①に示した data__n 回)
Lib_xcalibposition を呼び出し、最後に Lib_xcalibfinish を呼び出すことで変換係数の計算が終了します。

① ***order*** はキャリブレーションに使う方程式の最大次数です。
1～3 の範囲で指定してください。
この値を n ，元座標を (x, y)，変換後座標を (X, Y) とした時、座標変換に使われる方程式は次のようになります。

$$X = \sum_{j=1}^{n+1} \sum_{i=1}^{j} a_{ij} \cdot x^{i-1} \cdot y^{j-i}$$

$$Y = \sum_{j=1}^{n+1} \sum_{i=1}^{j} b_{ij} \cdot x^{i-1} \cdot y^{j-i}$$

ただし、 a_{ij} ：x 軸座標変換係数
b_{ij} ：y 軸座標変換係数

② ***data_n*** はキャリブレーションに使う基準点の総数です。内部で最小自乗法を使っているため、order の値によって最小値が決まっています。次の表を参考にしてください。

order	data_n の最小値(この値以上にしてください)
1	3
2	6
3	10

最大値はワークメモリの許す限り設定できます。できるだけ多くの座標を与えて計測誤差を補正してください。ワークメモリのサイズは以下の通り計算できます。
サイズ=32×data__n(バイト)

戻り値

処理結果

値	定　　数	意　　味
0	CLBERR_NOTHING	正常終了しました。
-1	CLBERR_NOT_ENOUGH_MARK	基準点が足りません。
-2	CLBERR_NOT_ENOUGH_MEMORY	ワークメモリが足りません。
-5	CLBERR_UNDEFINE_EQUATION	定義されていない方程式の次数を使おうとしています。

例

変換係数を決定し、元座標（15.0, 20.0）の変換後の座標を求めます。

```
#include  "f_calib.h"

void  main( void )
{
    struct  _xcoefficients  coeff;
    struct  _xcoord         original, conv;

    if( CLBERR_NOTHING == Lib_xcalibstart( 2, 6 ) )
    {
        if( CLBERR_NOTHING == Lib_xcalibposition( 45.0, -18.9, -20.2, -22.7 ) &&
            CLBERR_NOTHING == Lib_xcalibposition( 74.3,  16.3,  25.1, -14.5 ) &&
            CLBERR_NOTHING == Lib_xcalibposition( 44.5,  27.5,   9.8,  13.2 ) &&
            CLBERR_NOTHING == Lib_xcalibposition( 12.3,  13.0, -24.4,  23.0 ) &&
            CLBERR_NOTHING == Lib_xcalibposition( 58.4,  73.9,  49038.3 ) &&
            CLBERR_NOTHING == Lib_xcalibposition( 12.1,  68.7,  14.9,  62.0 ) )
        {
            if( CLBERR_NOTHING == Lib_xcalibfinish( &coeff ) )
            {
                original.x = 15.0;
                original.y = 20.0;

                if( CLBERR_NOTHING ==Lib_xcoordconv( &coeff, &original, &conv ))
                    Lib_printf( "( x, y )=( %lf, %lf )¥n¥r", conv.x, conv.y );
                else  Lib_printf( "Lib_xcoordconv error !¥n¥r" );
            }
            else  Lib_printf( "Lib_xcalibfinish error !¥n¥r" );
        }
        else  Lib_printf( "Lib_xcalibposition error !¥n¥r" );
    }
    else  Lib_printf( "Lib_xcalibstart error !¥n¥r" );
}
```

留意事項

○ 本ライブラリは、Lib_calibstart を高次方程式まで拡張したものです。
歪み補正等の非線形のキャリブレーションにお使いください。

○ 本ライブラリを複数個呼び出したら必ず、最後に Lib_xcalibfinish を呼び出してください。 さもないと、ワークメモリの開放がなされません。

Lib_xcalibposition

機　能

拡張キャリブレーションの対応点位置設定

形　式

```
#include   "f_calib.h"
int  Lib_xcalibposition( double  x, double  y, double  X, double  Y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

元の座標系の座標点(x,y)と、その点が対応する変換後座標系での座標位置(X,Y)を与え、ティーチングを実施します。ティーチングは相異なる基準点を複数点与えます。
(ただし、本ライブラリを呼び出せる最大回数は、Lib_xcalibstart で設定した値 data_n です。)
つまり、複数回にわたって本ライブラリを呼び出す必要があります。

① ***x*** は元座標 X 位置です。

② ***y*** は元座標 Y 位置です。

③ ***X*** は変換後座標 X 位置です。(出力される値ではないことに注意してください)

④ ***Y*** は変換後座標 Y 位置です。(出力される値ではないことに注意してください)

戻り値

処理結果

値	定　　数	意　　味
0	CLBERR_NOTHING	正常終了しました。
-3	CLBERR_TOO_MANY_POSITION	これ以上の基準点は登録できません。

「基準点とは何か」に関しては Lib__xcalibstart 内の解説を参照してください。

例

Lib_xcalibstart の例を参照してください。

留意事項

○ 本ライブラリは、Lib_calibposition を高次方程式まで拡張したものです。
　歪み補正等の非線形のキャリブレーションにお使いください。
○ 本ライブラリの呼び出し前に必ず Lib_xcalibstart を呼び出しておいてください。

Lib_xcalibfinish

機　能

拡張キャリブレーションの終了

形　式

```
#include  "f_calib.h"
int  Lib_xcalibfinish( struct _xcoefficients  *coefficient );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

一連のキャリブレーション操作の終了を通知します。
このライブラリにより、拡張座標変換のための変換係数を取得できます。

① ***coefficient*** は座標変換係数です。
構造体 _xcoefficients は f_calib.h 内で次のように宣言されています。

```
struct _xcoefficients
{
    int     n;        /* 使用した基準点の総数です    */
    double  kx[MAX_COEFF_N];/* x 軸座標変換係数     */
    double  ky[MAX_COEFF_N];/* y 軸座標変換係数     */
};
```

戻り値

処理結果

値	定　数	意　味
0	CLBERR_NOTHING	正常終了しました。
-4	CLBERR_DO_NOT_CALCULATE	与えられた基準点では計算できませんでした。

「基準点とは何か」に関しては Lib_xcalibstart 内の解説を参照してください。

例

Lib_xcalibstart の例を参照してください。

留意事項

○ 本ライブラリは、Lib_calibfinish を高次方程式まで拡張したものです。
　歪み補正等の非線形のキャリブレーションにお使いください。
○ 本ライブラリの呼び出し前に必ず Lib_xcalibstart を呼び出し、なおかつ、
　Lib_xcalibposition も複数回呼び出しておいてください。

Lib_xcoordconv

機　能

拡張座標変換の実施

形　式

```
#include   "f_calib.h"
int  Lib_xcoordconv( struct _xcoefficients  *coefficient,
                     struct _xcoord  *original, struct _xcoord  *conv );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

Lib_xcalibfinish で計算された座標変換係数を使い、座標変換を行います。

① ***coefficient*** は座標変換係数です。
Lib_xcalibfinish で取得したものをそのまま使います。
構造体 _xcoefficients は f_calib.h 内で次のように宣言されています。

```
struct _xcoefficients
{
    int         n;                  /* 使用した基準点の総数です */
    double  kx[MAX_COEFF_N];
    double  ky[MAX_COEFF_N];
};
```

② ***original*** は、元座標位置です。変換したい座標を入力してください。
構造体 _xcoord は f_calib.h 内で次のように宣言されています。

```
struct _xcoord
{
    double  x;        /* x 座標位置 */
    double  y;        /* y 座標位置 */
};
```

③ ***conv*** は、変換後座標位置です。元座標位置を座標変換した値が格納されます。
構造体 _xcoord に関しては項②を参照してください。

戻り値

処理結果（将来拡張用で、現行バージョンでエラーは返されません。）

値	定　数	意　味
0	CLBERR_NOTHING	正常終了しました。

例

Lib_xcalibstart の例を参照してください。

留意事項

○ 本ライブラリは、Lib_coordconv を高次方程式まで拡張したものです。
歪み補正等の非線形のキャリブレーションにお使いください。
○ 本ライブラリの呼び出し前に座標変換係数を確定しておく必要が有ります。
Lib_xcalibstart, Lib_xcalibposition, Lib_xcalibfinish を呼び出しておいてください。

19. カーソル関連ライブラリ

本ライブラリはカーソルに関するライブラリです。

Lib_init_cursor

機　能

カーソル初期化

形　式

```
#include "f_gui.h"
void Lib_init_cursor( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソル操作に必要な各種のパラメータを初期化します。
全てのカーソルの処理に先だって本ライブラリをコールしておいてください。

戻り値

ありません。

例

以下のサンプルは GUI1.C として CSC902¥sample(fv904¥sample)ディレクトリにインストールされています。

```
#include "f_stdio.h"
#include "f_gui.h"
#include "f_graph.h"

#define     NO_TOUCH      0    /* 下記の 2 つのキーとも押下なし  */
#define     EXECUTE       1    /* EXECUTE キーが入力された       */
#define     CANCEL        2    /* CANCEL キーが入力された        */
#define     ERASE         0    /* カーソルが消去中                */
#define     DRAW          1    /* カーソルが表示中                */

main()
{
    int  x_start, y_start;       /* 線分の始点        */
    int  x_cursor, y_cursor;     /* カーソル座標          */
    int  x_old, y_old;           /* 旧カーソル座標        */
    int  status;                 /* 実行/取消の情報   */

    /* カーソル初期化 */
    Lib_init_cursor();

    x_start = y_start = 0;
    x_cursor = x_old = 256;
    y_cursor = y_old = 240;

    Lib_drawline( x_start, y_start, x_cursor, y_cursor );

    /* カーソルを表示していないときはカーソル表示 */
     if( DRAW != Lib_get_cursor_mode() )
                                Lib_draw_cursor( x_cursor, y_cursor );

     for( status = NO_TOUCH; EXECUTE != status; )
     {                                  /* EXECUTE キーが押されるまで･･･ */
         /* カーソル情報取得 */
         status = Lib_see_current_position( &x_cursor, &y_cursor );

         if( x_cursor != x_old  ||  y_cursor != y_old )
         {                              /* 位置が変わったら表示を更新 */
             /* ｶｰｿﾙ移動 */
             Lib_move_cursor( x_cursor, y_cursor );
             Lib_erasline( x_start, y_start, x_old, y_old );
             Lib_drawline( x_start, y_start, x_cursor, y_cursor );

             x_old = x_cursor;
             y_old = y_cursor;
         }
     }
     Lib_printf(" set_point = ( %d, %d )¥r¥n",x_cursor,y_cursor);
     /* カーソル消去 */
     Lib_erase_cursor();

     Lib_erasline( x_start, y_start, x_old, y_old );
}
```

留意事項

ありません。

Lib_draw_cursor

機　能

カーソル表示

形　式

```
#include "f_gui.h"
void  Lib_draw_cursor( int x, int y );
```

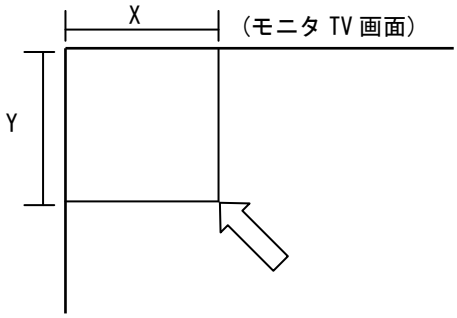
901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

モニタ画面上の与えられた座標位置にカーソルを表示します。

① x はカーソル X 座標(下図参照)

② y はカーソル Y 座標(下図参照)



戻り値

ありません。

例

Lib_init_cursor の例を参照してください。

留意事項

○ Lib_init_cursor を実行前にコールする必要があります。

FVL/LNX

○ 現在のところ本関数は何も実行しません。

Lib_erase_cursor

機　能

カーソル消去

形　式

```
#include "f_gui.h"
void  Lib_erase_cursor( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

モニタ画面上のカーソルを消去します。

戻り値

ありません。

例

Lib_init_cursor の例を参照してください。

留意事項

○ Lib_init_cursor を実行前にコールする必要があります。

FVL/LNX

○ 現在のところ本関数は何も実行しません。

Lib_move_cursor

機　能

カーソル移動

形　式

```
#include "f_gui.h"
void  Lib_move_cursor( int x, int y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソルを与えられた座標に移動します。移動前のカーソルは自動的に消去します。

① *x* は移動後のカーソル X 座標

② *y* は移動後のカーソル Y 座標

戻り値

ありません。

例

Lib_init_cursor の例を参照してください。

留意事項

○ Lib_init_cursor を実行前にコールする必要があります。

FVL/LNX

○ 現在のところ本関数は何も実行しません。

Lib_get_cursor_mode

機　能

カーソル表示モード取得

形　式

```
#include "f_gui.h"
int  Lib_get_cursor_mode( void );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

モニタ画面上に、カーソルが表示中か消去中かの状況を返します。

戻り値

値	定　　数	意　　味
0	ありません。	消去中です。
1	ありません。	表示中です。

例

Lib_init_cursor の例を参照してください。

留意事項

○ Lib_init_cursor を実行前にコールする必要があります。

Lib_see_current_position

機　能

カーソル情報取得

形　式

```
#include "f_gui.h"
int  Lib_see_current_position( int *x, int *y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソル位置座標および EXECUTE キーや CANCEL キーの入力状況を返します。

① *x はカーソル X 座標

② *y はカーソル Y 座標

戻り値

値	定　　数	意　　味
0	ありません。	下記の 2 つのキーとも押下されていません。
1	CURSOR_EXECUTE	EXECUTE キーが入力されました。
2	CURSOR_CANCEL	CANCEL キーが入力されました。

例

Lib_init_cursor の例を参照してください。

留意事項

○ Lib_look_current_position との違いは、以下の通りです。

```
/*-----------------------------------*/
int  x, y;

x = 100;  y = 200;
Lib_see_current_position( &x, &y );
/*-----------------------------------*/
```

と記述すると、カーソルが一旦、x=100 で、かつ、y=200 の位置に制御が移され、
この各々の x, y の値に移動量が加えられた値が返されます。

```
/*-----------------------------------*/
int  x, y;

x = 100;  y = 200;
Lib_look_current_position( &x, &y );
/*-----------------------------------*/
```

と記述すると、カーソルの現在位置を単に返すだけです。
つまり、ｘ＝100, ｙ＝200 の情報は何の意味も持ちません。

○ Lib_init_cursor を実行前にコールする必要があります。

Lib_look_current_position

機　能

カーソル情報取得Ⅱ

形　式

```
#include  "f_gui.h"
void Lib_look_current_position( int *x, int *y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソルの位置情報を返します。

① ***x*** はカーソルＸ座標位置です。

② ***y*** はカーソルＹ座標位置です。

戻り値

ありません。

例

```
#include      "f_stdio.h"
#include      "f_gui.h"
#include      "f_graph.h"

#define      NO_TOUCH       0    /* no input          */
#define      EXECUTE        1    /* input EXECUTE key */
#define      CANCEL         2    /* input CANCEL key  */
#define      ERASE          0    /* erase cursor      */
#define      DRAW           1    /* draw cursor       */

void  main()
{
    int  x_start, y_start;       /* starting point of line     */
    int  x_cursor, y_cursor;     /* cursor position            */
    int  x_old, y_old;           /* past cursor position       */
    int  status;                 /* execute/cancel information */

    /* initialize cursor */
    Lib_init_cursor();

    x_start = y_start = 0;
    Lib_look_current_position( &x_cursor, &y_cursor );
    x_old = x_cursor ;
    y_old = y_cursor;

    Lib_drawline( x_start, y_start, x_cursor, y_cursor );

    /* display cursor */
    if( DRAW != Lib_get_cursor_mode() )
                              Lib_draw_cursor( x_cursor, y_cursor );

    for( status = NO_TOUCH; EXECUTE != status; )
    {                                      /* while push EXECUTE key */
        /* get cursor information */
        status = Lib_see_current_position( &x_cursor, &y_cursor );

        if( x_cursor != x_old  ||  y_cursor != y_old )
        {       /* if the position changed then the display updated */
            /* move cursor */
            Lib_move_cursor( x_cursor, y_cursor );
            Lib_erasline( x_start, y_start, x_old, y_old );
            Lib_drawline( x_start, y_start, x_cursor, y_cursor );

            x_old = x_cursor;
            y_old = y_cursor;
        }
    }
    Lib_printf( " set point = ( %d, %d )¥r¥n", x_cursor, y_cursor );

    /* erase cursor */
    Lib_erase_cursor();

    Lib_erasline( x_start, y_start, x_old, y_old );
}
```

留意事項

○ Lib_see_current_position との違いは、以下の通りです。

```
/*-----------------------------------*/
int  x, y;

x = 100;  y = 200;
Lib_see_current_position( &x, &y );
/*-----------------------------------*/
```

と記述すると、カーソルが一旦、x=100 で、かつ、y=200 の位置に制御が移され、この各々の x,y のちに移動量が加えられた値が返されます。

```
/*-----------------------------------*/
int  x, y;

x = 100;  y = 200;
Lib_look_current_position( &x, &y );
/*-----------------------------------*/
```

と記述すると、カーソルの現在位置を単に返すだけです。
つまり、x=100,y=200 の情報は何の意味も持ちません。

○ Lib_init_cursor を実行前にコールする必要があります。

Lib_see_current_position_real

機　能

カーソル情報取得（現在状況取得）

形　式

```
#include "f_gui.h"
int Lib_see_current_position_real( int *x, int *y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソル位置座標および EXECUTE キーや CANCEL キーの現在状況を返します。
Lib_see_current_position との違いは Lib_see_current_position では EXECUTE キーや CANCEL キーの入力結果(トリガ)を返すのに対し、本ライブラリは EXECUTE キーや CANCEL キーの現在の押下状況を返します。

① ***x*** はカーソル X 座標

② ***y*** はカーソル Y 座標

戻り値

値	定　　数	意　　味
0	ありません	下記の 2 つのキーとも押下されていません。
1	CURSOR_EXECUTE	EXECUTE キーが押下されています。
2	CURSOR_CANCEL	CANCEL キーが押下されています。

例

カーソルキーの現在状況を取得します。

```
#include "f_gui.h"
void cursor()
{
    int x, y, status;

    Lib_init_cursor();
    x = 100; y = 200;
    status = Lib_see_current_position_real( &x, &y );
}
```

留意事項

○ Lib_init_cursor を実行前にコールする必要があります。

Lib_get_current_position

機　能

確定キー待ち機能付きカーソル情報取得

形　式

```
#include "f_gui.h"
int  Lib_get_current_position( int *x, int *y );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

EXECUTE キーもしくは CANCEL キーが入力されるまで待ち続け、これが入力された時点の位置を返します。

① *x はカーソル X 座標

② *y はカーソル Y 座標

戻り値

値	定　　数	意　　味
1	CURSOR_EXECUTE	EXECUTE キーが押下されました。
2	CURSOR_CANCEL	CANCEL キーが押下されました。

例

以下のサンプルは GUI2.C として CSC902¥sample(fv904¥sample)ディレクトリにインストールされています。

```
#include "f_stdio.h"
#include "f_gui.h"
#include "f_graph.h"
/* プロトタイプ */
void  draw_box( int, int, int, int );
void  erase_box( int, int, int, int );

#define      BOX_WIDTH          50        /* 探索領域の幅          */
#define      BOX_HEIGHT         50        /* 探索領域の高さ        */
#define      OFF                0         /* インアクティブ状態          */
#define      ON                 1         /* アクティブ状態            */
#define      EXECUTE            1         /* EXECUTE キーが押された */

main()
{
    int  xs, ys, xe, ye;          /* 領域の端点座標    */
    int  x_cursor, y_cursor;      /* カーソル座標          */
    int  process;                 /* ループ制御          */
    int  width, height;           /* 分解能            */

    width = Lib_get_fx_size();
    height = Lib_get_fy_size();

    /* カーソル初期化 */
    Lib_init_cursor();

    for( process = ON; ON == process; )
    {
        xs = rand() % ( width - BOX_WIDTH );
        ys = rand() % ( height - BOX_HEIGHT );
        xe = xs + BOX_WIDTH;
        ye = ys + BOX_HEIGHT;

        if( EXECUTE==Lib_get_current_position(&x_cursor,&y_cursor))
                                                        /*位置取得*/
        {                                              /* 実行指令のとき */
            if( xs <= x_cursor  &&  x_cursor <= xe  &&
                ys <= y_cursor  &&  y_cursor <= ye )
            {
                draw_box( xs, ys, xe, ye );
                Lib_printf( "O K" );
                erase_box( xs, ys, xe, ye );
            }
            else  Lib_printf( "N G" );
        }
        else  process = OFF;
    }
}

/* 枠線描画 */
void  draw_box( int xs, int ys, int xe, int ye )
{
```

```
    Lib_drawline( xs, ys, xe, ys );
    Lib_drawline( xe, ys, xe, ye );
    Lib_drawline( xs, ys, xs, ye );
    Lib_drawline( xs, ye, xe, ye );
}

/* 枠線消去 */
void  erase_box( int xs, int ys, int xe, int ye )
{
    Lib_erasline( xs, ys, xe, ys );
    Lib_erasline( xe, ys, xe, ye );
    Lib_erasline( xs, ys, xs, ye );
    Lib_erasline( xs, ye, xe, ye );
}
```

留意事項

○ Lib_init_cursor を実行前にコールする必要があります。

Lib_set_cursor_rate

機　能

カーソル移動速度率設定

形　式

```
#include "f_gui.h"
void  Lib_set_cursor_rate( int ratio );
```

901	902	903	904	FVL/LNX	高分解能
○	○	○	○	○	○

解　説

カーソルの移動速度を調整します。

① ***ratio*** は移動速度率です。標準速度を 100%として%で設定します。
設定範囲は 0～200 です。

戻り値

ありません。

例

カーソル初期化後に移動速度を 50%に設定します。

```
#include "f_gui.h"

void cursor_rate()
{
     Lib_init_cursor();
     Lib_set_cursor_rate( 50 );
}
```

留意事項

○ ユーザプログラムで設定した移動速度率は、ユーザプログラム終了時は、標準速度(100%)になります。

FVL/LNX

○ 現在のところ本関数は何も実行しません。

Lib_get_current_geometry

機　能

カーソル座標取得

形　式

```
#include "f_gui.h"
int  Lib_get_current_geometry( int *x, int *y );
```

901	902	903	904	FVL/LNX	高分解能
				○	

解　説

モニター画面左上を原点とするカーソルの位置を返します。

① *x はカーソル X 座標

② *y はカーソル Y 座標

戻り値

値	定　　数	意　　味
0	NORMAL_RETURN	正常終了しました。
1	ERROR_RETURN	異常終了しました。

付録. 各ライブラリの処理速度一覧

○計測対象機種

901	CSC901NT
902	FV902　model-3（Celeron　433MHz）
903	CSC903
904	FV904
FVL/LNX	FV2000

各ライブラリの実行所要時間欄の

⟶	空　白	使用不可
⟶	―――	条件により変化
⟶	・・・・・・	測定不能（所要時間が 0.001 ミリ秒未満）

を表しています。

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
1	Lib_adrs_bin_memory	0.814	0.001	・・・・・・	・・・・・・	・・・・・・	
2	Lib_adrs_gray_memory	0.814	0.001	・・・・・・	・・・・・・	・・・・・・	
3	Lib_alert_output	―――			―――		
4	Lib_alloc_bin_memory	1.628	0.051	1.628	0.814	・・・・・・	
5	Lib_alloc_gray_memory	1.628	0.051	1.817	0.814	1.000	
6	Lib_arc_box	34.18	0.382	58.268	2.441	0.400	
7	Lib_ascope	―――	―――	―――	―――	―――	
8	Lib_atan360	0.011	0.001	0.008	0.010	・・・・・・	
9	Lib_bin_memory_cls	1.628	0.057	1.763	0.814	0.020	
10	Lib_box	3.255	0.838	1.139	0.814	0.550	(0,0)-(511,479)
11	Lib_box_cls	11.393	11.393	15.381	4.069	4.400	512×480　線画
12	Lib_buff_to_memory	21.9/3.3	6.5/0.8	20.9/2.4	17.9/2.4	2.0/0.0	グレイ / 2値
13	Lib_calcline	0.814	0.006	0.244	0.027	0.002	4点
14	Lib_calibfinish	0.814	0.001	・・・・・・	・・・・・・	・・・・・・	
15	Lib_calibposition	0.081	0.001	0.081	・・・・・・	・・・・・・	
16	Lib_calibstart	0.027	0.001	・・・・・・	・・・・・・	・・・・・・	
17	Lib_cancetc	0.01	・・・・・・	0.006	・・・・・・	・・・・・・	
18	Lib_change_bin_memory	0.488	0.001	0.027	0.027	0.001	
19	Lib_change_gray_memory	0.461	0.001	0.027	0.027	0.001	
20	Lib_change_memory	0.949	0.003	0.054	0.054	0.001	

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
21	Lib_check_battery			0.736			
22	Lib_chrdisp	0.271	0.423	0.787	0.217	0.060	10 文字
23	Lib_circle	2.441	0.342	2.414	0.814	0.350	半径 240
24	Lib_close_clink	―――	―――	―――	―――		
25	Lib_close_cnet	―――	―――				
26	Lib_close_elink				―――	―――	
27	Lib_close_io	―――	―――	―――	―――	―――	
28	Lib_close_plink		―――				
29	Lib_clossio	―――	―――	―――	―――	―――	
30	Lib_clrcnetrcv	―――	―――				
31	Lib_cls	12.2/82.2	11.3/11.3	15.5/17.1	4.8/4.8	4.2/4.2	2 値 / グレイ
32	Lib_cnet_get_node_map	―――	―――				
33	Lib_cnet_irq_clear	―――	―――				
34	Lib_cnet_irq_set	―――	―――				
35	Lib_cnet_receive	―――	―――				
36	Lib_cnet_receive_status	―――	―――				
37	Lib_cnet_send	―――	―――				
38	Lib_cnet_send_status	―――	―――				
39	Lib_cnetbuffcheck	―――	―――				
40	Lib_cnetbuffclose	―――	―――				

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
41	Lib_cnetbuffclr	―――	―――				
42	Lib_cnetbuffopen	―――	―――				
43	Lib_cnetbuffrcv	―――	―――				
44	Lib_cnetbuffstatus	―――	―――				
45	Lib_coordconv	―――	―――			―――	
46	Lib_cosfunc	0.012	0.001	0.007	0.009	・・・・・・	
47	Lib_ddeciml	1.628	0.293	0.732	0.244	0.040	6文字
48	Lib_display_comment		0.391	―――	―――	0.180	
49	Lib_display_histogram	―――	―――	―――	―――	―――	
50	Lib_dispval	0.217	0.122	0.326	0.109	0.020	3文字
51	Lib_draw_cursor	1.085	0.001	0.678	0.217	・・・・・・	
52	Lib_drawcircle	3.255	0.342	2.414	0.814	0.350	半径240
53	Lib_drawline	1.628	0.130	0.732	0.163	0.140	(0,0)-(511,479)
54	Lib_drawquad		・・・・・・	38.248	4.8828		
55	Lib_elink_client				―――	―――	
56	Lib_elink_disconnect				―――	―――	
57	Lib_elink_get_client				―――	―――	
58	Lib_elink_receive				―――	―――	
59	Lib_elink_send				―――	―――	
60	Lib_elink_server				4.882	―――	

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
61	Lib_elink_status				.....	.....	
62	Lib_enlarge	44.759	4.883	28.483	7.324	1.200	
63	Lib_erascircle	3.255	0.342	2.387	0.814	0.350	半径 240
64	Lib_erase_cursor	0.027	0.001	0.027	0.027	.....	
65	Lib_erasline	1.628	0.130	0.705	0.380	0.140	(0,0)-(511,479)
66	Lib_ex_ascope	———	———	———	———	———	
67	Lib_external_input	0.163			.....		
68	Lib_external_output	0.163			.....		
69	Lib_external_trigger	0.081		0.027	0.027		
70	Lib_external_get_exposure_output				0.050		
71	Lib_external_set_exposure_output				0.050		
72	Lib_fcalcircl	1.628	0.031	1.329	0.027	0.002	8 点
73	Lib_fcalelps	4.069	0.071	6.348	0.814	0.042	16 点
74	Lib_fcalquad		.....	4.8828	0.8138	.....	
75	Lib_free	0.353	0.001	0.19	0.027	0.007	alloc + free
76	Lib_free_bin_memory	2.306	0.050	1.628	0.434	0.026	alloc + free
77	Lib_free_gray_memory	2.333	0.051	1.817	0.488	0.326	alloc + free
78	Lib_g3dwire	———	———	———	———	———	
79	Lib_get	20.345	7.324	31.087	8.952	10.000	全面
80	Lib_get_adrs_work_memory	0.814	0.001				

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
81	Lib_get_bin_memory	0.027	------	------	------	------	
82	Lib_get_bin_memory_size					------	
83	Lib_get_current_position	———	———	———	———	———	
84	Lib_get_cursor_mode	0.027	0.001	------	------	------	
85	Lib_get_date		------	0.736		------	
86	Lib_get_dx_size	0.027	------	------	------	------	
87	Lib_get_dy_size	0.027	------	------	------	------	
88	Lib_get_font_type	0.023	------	0.011	0.012	------	
89	Lib_get_fx_size	0.027	------	------	------	------	
90	Lib_get_fy_size	0.027	------	------	------	------	
91	Lib_get_gray_memory	------	------	0.003	------	------	
92	Lib_get_gray_memory_size					------	
93	Lib_get_io_error	------	------	0.006	------	------	
94	Lib_get_io_timeout	------	------	0.006	------	------	
95	Lib_get_memory_type	0.027	------	------	------	------	
96	Lib_get_time		0.012	0.736		------	
97	Lib_getc_clink	———	———	———	———		
98	Lib_getc_cnet	———	———				
99	Lib_getc_plink		———				
100	Lib_getc_rs232c	———	———	———	———	———	

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
101	Lib_getchar	―――	―――	―――	―――	―――	
102	Lib_getcnetrcv	―――	―――				
103	Lib_gets	―――	―――	―――	―――	―――	
104	Lib_gets_clink	―――	―――	―――	―――		
105	Lib_gets_cnet	―――	―――				
106	Lib_gets_plink	―――	―――				
107	Lib_getsi	―――	―――	―――	―――	―――	
108	Lib_gray_memory_cls	10.579	4.883	13.428	4.069	1.000	
109	Lib_gray_transmit	36.621	4.883	28.157	12.207	6.000	上半分
110	Lib_histogram	65.104	4.883	41.802	11.393	3.000	
111	Lib_histogram_step	23.600	2.441	12.234	3.255	1.000	
112	Lib_init_cursor	0.380	0.019	0.054	0.027	・・・・・・	
113	Lib_invdisp	0.244	0.553	0.76	0.217	0.080	10 文字
114	Lib_io_buff_check	―――	―――	―――	―――	―――	
115	Lib_kanjishift	0.163	0.423	0.76	0.217	0.060	10 文字
116	Lib_lfree	0.217	0.005	0.027	0.027	0.015	alloc + free
117	Lib_line	1.628	0.130	0.705	0.163	0.140	(0,0)-(511,479)
118	Lib_look_current_position	0.027	0.001	・・・・・・	・・・・・・	・・・・・・	
119	Lib_malloc	0.353	0.005	0.19	0.027	0.007	alloc + free
120	Lib_memory_clear	13.0/92.7	11.0/15	17.3/30.6	4.8/8.1	4.4/5.8	2 値 / グレイ

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
121	Lib_mlalloc	0.353	0.005	0.027	0.027	0.015	alloc + free
122	Lib_move_cursor	1.628	0.001	0.027	0.027	.....	
123	Lib_open_clink	———	———	———	———		
124	Lib_open_cnet	———	———				
125	Lib_open_elink				0.048	———	
126	Lib_open_io	———	———	———	———	———	
127	Lib_open_plink		———				
128	Lib_opensio	———	———	———	———	———	
129	Lib_paint	808.103	82.194	413.41	203.450	207.000	256×256
130	Lib_paint2	794.269	65.104	362.955	189.615	186.000	256×256
131	Lib_plot	4.883	0.562	2.767	0.814	0.620	25 点
132	Lib_pol_to_rec	136.718	8.952	171.061	65.104	2.000	例にある条件
133	Lib_printf	———	———	———	———	———	
134	Lib_pset	341.796	2.441	125.325	133.463	0.020	10000 点（FVL/LNX は 1000 点）
135	Lib_put	17.904	13.835	24.495	8.952	15.000	全面
136	Lib_putc_clink	———	———	———	———		
137	Lib_putc_cnet	———	———				
138	Lib_putc_plink		———				
139	Lib_putc_rs232c	———	———	———	———	———	
140	Lib_putchar	———	———	———	———	———	

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
141	Lib_puts	―――	―――	―――	―――	―――	
142	Lib_puts_clink	―――	―――	―――	―――		
143	Lib_puts_cnet	―――	―――				
144	Lib_puts_plink		―――				
145	Lib_putso	―――	―――	―――	―――	―――	
146	Lib_readclk	―――	―――	―――	―――	―――	
147	Lib_readclk_count	―――	―――	―――	―――		
148	Lib_receive_status_plink		―――				
149	Lib_recover_plane	17.904	13.835	24.658	8.952	15.400	
150	Lib_reduce	44.759	5.697	38.900	9.766	2.000	
151	Lib_rotation	209.960	15.462	174.967	60.221	6.000	(256,240)中心に 30°
152	Lib_scale	346.679	5.697	73.893	26.042	2.000	2 倍
153	Lib_scanf	―――	―――	―――	―――	―――	
154	Lib_see_current_position	0.434	0.004	0.163	0.027	19.967	
155	Lib_see_current_position_real	0.434	0.004	0.163	0.027	19.967	
156	Lib_sensetc	―――	―――	―――	―――	―――	
157	Lib_set_cursor_rate	.....	.....	.....	.....	.....	
158	Lib_set_cursor_shape	0.163	11.393	0.030	.....	.....	
159	Lib_set_date		.....	1.489		.....	
160	Lib_set_font_type	0.062	0.001	0.029	0.024	.....	

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
161	Lib_set_io_timeout	------	------	0.006	------	------	
162	Lib_set_time		------	1.472		------	
163	Lib_setcnetrcv	———	———				
164	Lib_shift	48.014	7.324	39.062	12.207	3.000	50 画素シフト
165	Lib_sinfunc	0.010	0.001	0.007	0.008	------	
166	Lib_speed	———	———	———	———	———	
167	Lib_shelter_plane	30.924	17.904	46.712	13.835	27.000	
168	Lib_sprintf	———	———	———	———	———	1 文字 / 100 文字
169	Lib_sqrt32	0.012	0.001	0.008	0.009	------	
170	Lib_sscanf	———	———	———	———	———	
171	Lib_stopclk_count	———	———	———	———	———	
172	Lib_strtclk	———	———	———	———	———	
173	Lib_strtclk_count	———	———	———	———	———	
174	Lib_strtetc	———	———	———	———	———	
175	Lib_time_delay	———	———	———	———	———	
176	Lib_time_disp	———	———	———	———	———	
177	Lib_view	11.393	11.393	15.381	4.069	13.000	(0,0)-(511,479)
178	Lib_vout_create_video_window					216.000	
179	Lib_vout_destroy_video_window					28.000	
180	Lib_vout_get_color					------	

基本ライブラリ実行所要時間代表値一覧

単位　　ミリ秒

項番	ライブラリ名	所要時間					備　　考
		901	902	903	904	FVL/LNX	
181	Lib_vout_get_video_no					.....	
182	Lib_vout_move_video_window					.....	
183	Lib_vout_set_color					.....	
184	Lib_vout_set_video_no					.....	
185	Lib_vout_sync_framebuffer					———	
186	Lib_xbcircle	1.248	0.114	0.868	0.814	0.070	半径 100
187	Lib_xbline	1.194	0.114	0.868	0.814	0.070	(0,0)-(511,479)
188	Lib_xcalibfinish	4.883	0.051	8.87	0.814	0.076	Order･･･2,　data_n･･･6
189	Lib_xcalibposition	———	———	———	———	———	
190	Lib_xcalibstart	0.814	0.051	8.843	0.651	0.077	Order･･･2,　data_n･･･6
191	Lib_xcoordconv	0.081	0.002	0.163	.....	———	
192	Lib_xgline	0.570	0.033	0.515	0.163	0.080	(0,0)-(511,479)
193	Lib_xreadclk					———	
194	Lib_xrotation	761.717	32.552	460.285	167.643	13.000	
195	Lib_xscale	740.558	39.551	283.853	100.097	7.000	
196	Lib_xgcircle	0.543	0.024	0.488	0.217	0.070	半径 100
197	Lib_zbcircle	1.628	0.114	0.895	0.814	0.070	半径 100
198	Lib_zbline	1.194	0.114	0.895	0.814	0.070	(0,0)-(511,479)
199	Lib_zgcircle	1.628	0.024	0.57	0.244	0.070	半径 100
200	Lib_zgline	0.678	0.033	0.624	0.163	0.070	(0,0)-(511,479)

索　引

A

Lib_adrs_bin_memory 93
Lib_adrs_gray_memory 92
Lib_alert_output 284
Lib_alloc_bin_memory 90
Lib_alloc_gray_memory 88
Lib_arc_box 65
Lib_ascope 137
Lib_atan360 317

B

Lib_bin_memory_cls 125
Lib_box 27
Lib_box_cls 33
Lib_buff_to_memory 118

C

Lib_calcline 322
Lib_calibfinish 356
Lib_calibposition 355
Lib_calibstart 353
Lib_cancetc 298
Lib_change_bin_memory 103
Lib_change_gray_memory 102
Lib_change_memory 101
Lib_check_battery 309
Lib_chrdisp 14
Lib_circle 31
Lib_close_clink 242
Lib_close_cnet 192
Lib_close_elink 266
Lib_close_io 156
Lib_close_plink 254
Lib_clossio 186
Lib_clrcnetrcv 217
Lib_cls 35
Lib_cnet_get_node_map 196
Lib_cnet_irq_clear 195
Lib_cnet_irq_set 193
Lib_cnet_receive 200
Lib_cnet_receive_status 198
Lib_cnet_send 203
Lib_cnet_send_status 201
Lib_cnetbuffcheck 221
Lib_cnetbuffclose 215
Lib_cnetbuffclr 223
Lib_cnetbuffopen 214
Lib_cnetbuffrcv 222
Lib_cnetbuffstatus 225
Lib_coordconv 358
Lib_cosfunc 315

D

Lib_ddeciml 20
Lib_display_comment 13
Lib_display_histogram 134
Lib_dispval 18
Lib_draw_cursor 370
Lib_drawcircle 29
Lib_drawline 25
Lib_drawquad 332

E

Lib_elink_client 268
Lib_elink_disconnect 274
Lib_elink_get_client 273
Lib_elink_receive 271
Lib_elink_send 272
Lib_elink_server 267
Lib_elink_status 270
Lib_enlarge 343
Lib_erascircle 30
Lib_erase_cursor 371
Lib_erasline 26
Lib_ex_ascope 144
Lib_external_get_exposure_output 288
Lib_external_input 282
Lib_external_output 283
Lib_external_set_exposure_output 286
Lib_external_trigger 285

F

Lib_fcalcircl 325
Lib_fcalelps 328
Lib_fcalquad 331
Lib_free 112
Lib_free_bin_memory 97
Lib_free_gray_memory 95

G

Lib_g3dwire 139
Lib_get 37

索　引

Lib_get_adrs_work_memory 115
Lib_get_bin_memory 100
Lib_get_bin_memory_size 120
Lib_get_current_geometry 384
Lib_get_current_position 380
Lib_get_cursor_mode 373
Lib_get_date 306
Lib_get_dx_size 108
Lib_get_dy_size 109
Lib_get_font_type 9
Lib_get_fx_size 104
Lib_get_fy_size 106
Lib_get_gray_memory 99
Lib_get_gray_memory_size 119
Lib_get_io_error 171
Lib_get_io_timeout 152
Lib_get_memory_type 114
Lib_get_time 304
Lib_getc_clink 244
Lib_getc_cnet 210
Lib_getc_plink 257
Lib_getc_rs232c 184
Lib_getchar 158
Lib_getcnetrcv 219
Lib_gets 160
Lib_gets_clink 243
Lib_gets_cnet 208
Lib_gets_plink 256
Lib_getsi 181
Lib_gray_memory_cls 126
Lib_gray_transmit 117

H

Lib_histogram 130
Lib_histogram_step 132

I

Lib_init_cursor 368
Lib_invdisp 16
Lib_io_buff_check 172

K

Lib_kanjishift 11

L

Lib_lfree 113
Lib_line 23
Lib_look_current_position 376

M

Lib_malloc 110
Lib_memory_clear 124
Lib_mlalloc 111
Lib_move_cursor 372

O

Lib_open_clink 240
Lib_open_cnet 190
Lib_open_elink 265
Lib_open_io 154
Lib_open_plink 252
Lib_opensio 176

P

Lib_paint 45
Lib_paint2 47
Lib_plot 49
Lib_pol_to_rec 347
Lib_printf 163
Lib_pset 22
Lib_put 39
Lib_putc_clink 247
Lib_putc_cnet 213
Lib_putc_plink 260
Lib_putc_rs232c 185
Lib_putchar 159
Lib_puts 162
Lib_puts_clink 246
Lib_puts_cnet 211
Lib_puts_plink 259
Lib_putso 183

R

Lib_readclk 295
Lib_readclk_count 301
Lib_receive_status_plink 255
Lib_recover_plane 43
Lib_reduce 344
Lib_rotation 335

S

Lib_scale 339
Lib_scanf 167

索　引

Lib_see_current_position 374
Lib_see_current_position_real..... 379
Lib_sensetc 297
Lib_set_cursor_rate 383
Lib_set_cursor_shape 10
Lib_set_date 308
Lib_set_font_type 8
Lib_set_io_timeout 150
Lib_set_time 305
Lib_setcnetrcv 216
Lib_shelter_plane 41
Lib_shift 345
Lib_sinfunc 314
Lib_speed 178
Lib_sprintf 166
Lib_sqrt32 316
Lib_sscanf 170
Lib_stopclk_count 303
Lib_strtclk 294
Lib_strtclk_count 300
Lib_strtetc 296

T

Lib_time_delay 299
Lib_time_disp 302

V

Lib_view 44
Lib_vout_create_video_window..... 67
Lib_vout_destroy_video_window..... 69
Lib_vout_draw_video_window..... 80
Lib_vout_erase_video_window 81
Lib_vout_get_color 72
Lib_vout_get_geometry 75
Lib_vout_get_option 82
Lib_vout_get_resolution 76
Lib_vout_get_video_no 70
Lib_vout_move_video_window 74
Lib_vout_set_color 73
Lib_vout_set_option 83
Lib_vout_set_video_no 71
Lib_vout_set_video_window_title 79
Lib_vout_sync_framebuffer 77

X

Lib_xbcircle 54
Lib_xbline 50
Lib_xcalibfinish 363
Lib_xcalibposition 362
Lib_xcalibstart 360
Lib_xcoordconv 364
Lib_xgcircle 61
Lib_xgline 57
Lib_xreadclk 310
Lib_xrotation 337
Lib_xscale 341

Z

Lib_zbcircle 56
Lib_zbline 52
Lib_zgcircle 63
Lib_zgline 59

90X　基本ライブラリ説明書　*Vol.2*

2009 年 4 月第 14 版発行

発行所　　株式会社ファースト

本　　社　〒242-0001　神奈川県大和市下鶴間 2791-5

ユーザ・サポート　　FAX　046-272-8692
E-mail : support@fast-corp.co.jp

B-000894